TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

VIZUALIZACE DAT SPOTŘEBY ELEKTRICKÉ ENERGIE PROSTŘEDNICTVÍM WEBOVÉ APLIKACE

Bakalářská práce

Studijní program: B2646 – Informační technologie
Studijní obor: 1802R007 – Informační technologie

Autor práce: **Zdeněk Rindt**
Vedoucí práce: Ing. Jan Kraus, Ph.D.

Liberec 2014

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

VISUALIZATION OF ELECTRICITY CONSUMPTION IN A WEB-BASED APPLICATION

Bachelor thesis

Study programme: B2646 – Information technology
Study branch: 1802R007 – Information technology

Author: **Zdeněk Rindt**
Supervisor: Ing. Jan Kraus, Ph.D.

Liberec 2014

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: **2013/2014**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Zdeněk Rindt**
Osobní číslo:	**M10000185**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Vizualizace dat spotřeby elektrické energie prostřednictvím webové aplikace**
Zadávající katedra:	**Ústav mechatroniky a technické informatiky**

Zásady pro vypracování:

1. Seznamte se se strukturou databáze ENVIS a s navrženými rozhraními pro přístup k uloženým datům.
2. Seznamte se s trendy ve vývoji webových aplikací s přívětivým uživatelským rozhraním a grafickými prvky, zaměřte se na existující webové aplikace pro tzv. energetický management.
3. Navrhněte vlastní aplikaci pro zobrazování a základní analýzu uložených dat spotřeby energií.
4. Shrňte dosažené výsledky a diskutujte další možnosti rozvoje tématu.

Rozsah grafických prací: **dle potřeby dokumentace**

Rozsah pracovní zprávy: **30–40 stran**

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

[1] **MICROSOFT CORPORATION. MSDN Library [online]. 2013 [cit. 2013-09-23]. Dostupné z: http://msdn.microsoft.com/**

[2] **MICROSOFT CORPORATION. Microsoft Application Architecture Guide, 2nd Edition [online]. 2009 [cit. 2013-09-23]. Dostupné z: http://msdn.microsoft.com/en-us/library/ff650706.aspx**

[3] **W3C. HTML5: HTML5 A vocabulary and associated APIs for HTML and XHTML [online]. 2012 [cit. 2013-09-23]. Dostupné z: http://www.w3.org/TR/html5/**

[4] **MOZILLA DEVELOPER NETWORK. Javascript [online]. 2013 [cit. 2013-09-23]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript**

Vedoucí bakalářské práce: **Ing. Jan Kraus, Ph.D.**
Ústav mechatroniky a technické informatiky

Datum zadání bakalářské práce: **10. října 2013**

Termín odevzdání bakalářské práce: **16. května 2014**

prof. Ing. Václav Kopecký, CSc.
děkan

L.S.

doc. Ing. Milan Kolář, CSc.
vedoucí ústavu

V Liberci dne 10. října 2013

Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Datum:

Podpis:

Abstrakt

Cílem práce je návrh a implementace webové aplikace pro základní analýzu a vizualizaci dat spotřeby elektrické energie, naměřených přístroji společnosti KMB systems s. r. o. a uložených v databázi Microsoft SQL Server, generované softwarem ENVIS, a to v pokud možno uživatelsky přívětivém rozhraní.

Aplikace je postavena na platformě .NET Framework pro webový server IIS a skládá se ze serverové části, která poskytuje data pomocí rozhraní API a klientské části, která toto rozhraní využívá.

Klíčová slova

Webová aplikace, .NET Framework, ASP.NET, Microsoft SQL Server, Entity Framework, JavaScript, Single Page Application, energetický management, MVVM, ENVIS

Abstract

This paper focuses on design and implementation of a web application for basic analysis and visualization of electricity consumption data, measured by electric meters made by KMB systems s. r. o. and stored in a Microsoft SQL Server database generated by ENVIS.

The application can run on an IIS web server as a .NET Framework application and consists of a server part, providing all kind of data stored in the database via API, and a client part, which uses this API to retrieve, process and visualize these data.

Keywords

Web Application, .NET Framework, ASP.NET, Microsoft SQL Server, Entity Framework, JavaScript, Single Page Application, energy management, MVVM, ENVIS

Poděkování

Rád bych poděkoval Ing. Janu Krausovi, Ph.D. za jeho vedení a konzultace a Ing. Pavlu Štěpánovi za technické konzultace týkající se pozadí software ENVIS. Zejména bych pak rád poděkoval mé rodině a všem svým přátelům za jejich podporu.

Obsah

Seznam obrázků

Seznam tabulek

Seznam zkratek

AMD	Asynchronous Module Definition
API	Application Programmable Interface
ASP	Active Server Pages
CLR	Common Language Runtime
CSS	Cascading Style Sheets
DOM	Document Object Model je JavaScript objekt reprezentující HTML dokument
EF	Entity Framework
FM	Fakulta mechatroniky, informatiky a mezioborových studií Technické univerzity v Liberci
HTML	Hypertext Markup Language, nebo-li hypertextový značkovací jazyk
HTTP	Hypertext Transport Protocol, je bezstavový textový protokol pro přenos informací používaný v prostředí webu
IIS	Internet Information Service, je webový server vyvíjený společností Microsoft pro platformu Windows
JS	JavaScript
JSON	JavaScript Object Notation, formát umožňující ukládat data ve formě JavaScript objektu
LINQ	Language Integrated Query
MVC	Model View Controller
MVVM	Model View ViewModel
OData	Open Data, protokol rozhraní API definující způsob dotazování nad entitami
ORM	Object-Relational Mapping
SPA	Single-page Application, je webová aplikace sestávající z jediné stránky, která se transformuje v průběhu práce s aplikací bez nutnosti viditelného znovunačtení ze serveru
SQL	Structured Query Language, je jazyk používaný relačními databázovými systémy pro vytváření dotazů k výběru, vložení, modifikaci, nebo odstranění dat, nebo prvků schéma databáze
SSD	Solid-state Drive
TUL	Technická univerzita v Liberci

1. Úvod

Cílem této bakalářské práce je vývoj jednoduché webové aplikace pro energetický management. Tato webová aplikace by měla uživateli v přívětivém grafickém rozhraní umožnit základní analýzu a porovnání dat spotřeby elektrické energie sesbíraných pomocí specializovaných měřicích přístrojů.

Zadání práce pochází ze společnosti KMB systems s. r. o., která vyvíjí vlastní produkty pro kompenzaci jalového výkonu, univerzální měřicí přístroje, analyzátory kvality elektrické energie a software pro analýzu měřených dat, ENVIS.

1.1 Měřicí přístroje pro energetický management

Elektroměry měřicích přístrojů společnosti KMB systems s. r. o. v době psaní textu umožňují dle manuálu (KMB systems, s. r. o., 2013, str. 6) čtyřkvadrantní měření elektrické energie a registraci ve třech tarifních pásmech, a to jak jednofázových, tak třífázových hodnot, dále záznam maxim průměrných činných výkonů za současný a předchozí měsíc a celkového dosaženého maxima a automatickou registraci hodnot elektroměru s nastavitelnou periodou odečtu.

Přístroje jsou vybaveny rozhraním USB pro přenos dat, nastavení přístroje a pro upgrade firmware a volitelně také rozhraním RS-232, RS-485, nebo Ethernet. Pomocí software ENVIS je možné s přístrojem komunikovat a na něm uložená data vizualizovat, nebo archivovat.

1.2 Měření

V místě zákazníka může být instalováno hned několik měřicích přístrojů, které sbírají data o kvalitě odebírané energie a spotřebě. Modelovým příkladem může být hlavní měřicí přístroj pro budovu a podružné měřicí přístroje pro každé patro, případně oddělení. Tímto způsobem je možné monitorovat spotřebu v jednotlivých částech objektu a zjistit, které oddělení se jakým způsobem podílí na celkové spotřebě elektrické energie.

Díky podružnému měření je možné nejen měřit spotřebu jednotlivých pater, nebo oddělení, ale provádět také analýzu ztrát, nebo-li měřit rozdíl mezi nadřazeným a podružnými měřeními.

Dalším důvodem pro instalaci měřicích přístrojů a analyzátorů kvality je pro firemní zákazníky kompenzace jalového výkonu, která se však tématu práce netýká.

1.3 Čtvrthodinové maximum

Co se naopak energetického managementu z pohledu firemního zákazníka týká, je tzv. čtvrthodinové maximum, které omezuje možnou spotřebu ve kteroukoliv čtvrthodinu dne na sjednané maximum. Překročení sjednaného maxima je ze strany dodavatele penalizováno, avšak pro výjimečné překročení se dle Majda (2008) nevyplatí ani sjednání maxima s rezervou.

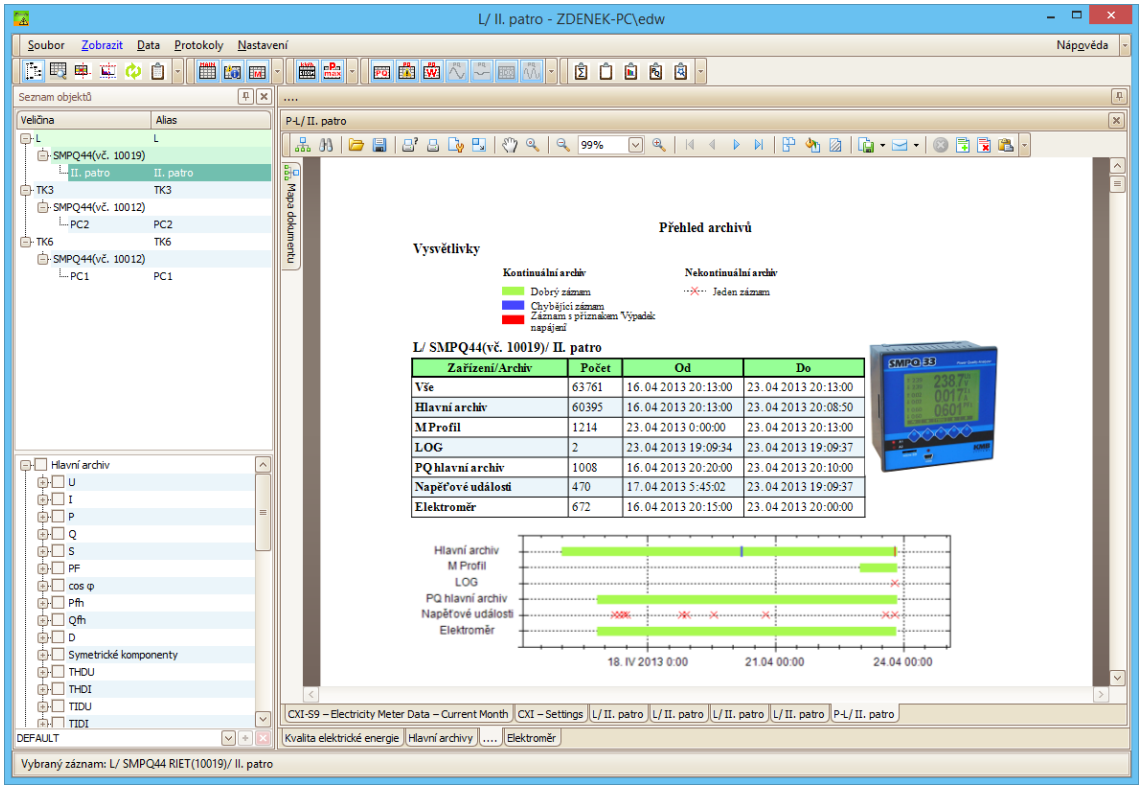
Cílem čtvrthodinového maxima je proto pokud možno rovnoměrná spotřeba v průběhu dne, k čemuž se využívá regulátorů maxima, které dokáží zátěž odpojovat v případě, že hrozí překročení stanoveného maxima. Inteligentní systémy pak dokáží připojovat a odpojovat méně prioritní zátěž, případně řídit celé linky. Důležitá je také organizace práce, jak píše Majda (2008), tedy nevyužívání některých spotřebičů v době zvýšené zátěže, například kvůli topení.

1.4 Vizualizační software ENVIS

Společnost KMB systems s. r. o. nabízí zdarma ke stažení software ENVIS, který umožňuje správu jednotlivých měřicích přístrojů a on-line sledování jejich stavu, stejně jako archivaci na nich uložených dat. Ta je možné uložit do souboru **.CEA*, případně importovat do databáze Microsoft SQL Server.

Obrázek 1.1: Část okna aplikace ENVIS

Dle manuálu (KMB systems s.r.o, 2012) patří mezi funkce software například možnost sledovat napěťové události, jako je podpětí, přepětí, případně i výpadky (umožňuje-li to přístroj), prohlížet záznamy o kvalitě energie, například napětí, proud, frekvenci, účiník, atd. K dispozici jsou také data elektroměru (viz obrázek 1.1). Aplikace umožňuje vizualizovat všechny záznamy ve formě tabulek, nebo grafů, vyhodnocovat kvalitu energie podle normy EN 50160, nebo generovat protokoly (viz obrázek 1.2).



Obrázek 1.2: Celkový protokol vygenerovaný pomocí ENVIS

2. Webové aplikace

Jako mnoho jiných technologií, i projekt webu (viz Berners-Lee – Cailliau (1990)) se od svého vzniku v CERN na počátku 90. let hodně změnil, ačkoliv jeho základ zůstal stejný. Bezstavový protokol HTTP pro komunikaci mezi webovým prohlížečem a serverem byl rozšířen. Jednoduchý jazyk HTML, definující strukturu dokumentu, doplnily kaskádové styly CSS, které dokumentu dávají formu a skriptovací jazyk JavaScript, pomocí kterého lze reagovat na události v dokumentu a provádět modifikace dokumentu na straně webového prohlížeče.

Moderní webové aplikace už nejsou jen jednoduché webové stránky, ale komplexní aplikace, které si svou složitostí nezadají s aplikacemi nativními. Některé webové aplikace proto vsází na uživatelské rozhraní podobné nativním aplikacím, například on-line kancelář Google Docs, nebo konkurenční Office 365, které mohou být v terénu dostatečnou náhradou za nativní kancelářský balík.

Tyto aplikace se vyznačují tím, že od chvíle jejich načtení nedochází při práci ke znovunačítání stránky s kliknutím na každý ovládací prvek, ale veškerá komunikace s webovým serverem probíhá na pozadí a stránka aplikace se transformuje dynamicky. Těmto aplikacím se pak říká client-side aplikace, nebo také single-page aplikace (dále jen SPA).

2.1 Moderní trendy

Jak jsem uvedl v předchozím odstavci, moderní aplikace mohou být velice komplexní programy využívající hned několik technologií a jazyků. Prim hraje především JavaScript, který umožňuje pracovat s DOM dokumentu a reagovat na události v něm, nebo v prohlížeči. Má přístup k úložišti Local Storage, pokud jej prohlížeč uživatele podporuje, Cookies a umožňuje také programování 3D grafiky a her. Technologie WebSockets umožňuje klientské aplikaci otevřít dlouhodobé spojení se vzdáleným serverem a využívat jej k výměně informací, čehož donedávna nebylo možné bez použití technologie, jako je Flash.

2.1.1 JavaScript

Jelikož jsou podobné aplikacím nativním, dostávají se u webových aplikací ke slovu návrhové vzory a architektury nativních aplikací, například návrhový vzor MVC, případně MVVM. Aby nemusel programátor začínat na zelené louce, existují frameworky, které vývoj dynamických webových aplikací s využitím některého z návr-

hových vzorů usnadní, například Angular, který je zástupcem MVW[1], Ember, nebo Durandal.

Tyto frameworky navíc řeší další problémy spojené s vývojem SPA, například data binding mezi pohledem (View) a ViewModelem. Pohled je představován HTML šablonou, přičemž se nemusí jednat o kompletní dokument, ale třeba jen část stránky, nebo dokonce element, a do této šablony jsou vkládána data uložená v Modelu. Vytvoříme ViewModel, který reprezentuje data pohledu, načteme do něj data z Modelu a deklarujeme v pohledu propojení s ViewModelem. Framework poté zajistí, že data z ViewModelu se automaticky vloží do pohledu a naopak, pokud uživatel modifikuje pohled, data ViewModelu se aktualizují daty pohledu.

2.1.2 HTML5

Úzce s JavaScriptem souvisí také HTML5, které je synonymem pro moderní web a přináší mnoho nových technologií, z nichž část jsem již zmínil v předešlých řádcích. Ve zkratce se na ně podívejme:

CSS3 nová revize kaskádových stylů přidávající podporu pro animace, 2D a 3D transformace, textové efekty, vícesloupcový layout, nebo barevné přechody a další.

Média elementy pro přehrávač audia a videa bez nutnost vkládání Flashového přehrávače.

Validace validování formulářů na základě deklarovaného typu vstupních hodnot prvků už na straně prohlížeče.

Canvas element pro vykreslování grafiky pomocí JavaScript API.

History API umožňuje manipulaci s historií prohlížeče u aplikací typu SPA, kde prohlížeč bez pomoci programátora nedokáže zaznamenávat historii pohybu v aplikaci, která běží v rámci jedné jediné stránky.

Web Sockets umožňuje otevřít trvalé spojení mezi aplikací a serverem pro výměnu dat.

Local Storage je úložiště v prohlížeči typu klíč-hodnota pro data aplikace.

Tento seznam samozřejmě není kompletní a jedná se pouze o výčet zajímavých technologií a API, které HTML5, nacházející se stále ve fázi kandidát na vydání, přináší. Specifikace HTML5 je umístěna přímo na webu organizace W3C[2]. Seznam technologií byl čerpán z Mozilla Developer Network (2014).

[1] Angular zavádí Model a View, na programátorovi je, jestli použije Controller, nebo ViewModel, případně jiný vzor, který jemu vyhovuje.

[2] Dostupné on-line z: `http://www.w3.org/TR/html5/`

2.2 Energetický management ve webových aplikacích

Snaha o optimalizaci nákladů roste spolu s cenou energií, a proto se na trh dostává čím dál více společností a firem, které se energetickým management zabývají, ať už přímo, nebo okrajově. Některé ze společností nejenže vyvíjí aplikace samotné, ale také vlastní měřící zařízení. Pro účely srovnání uvádím několik webových aplikací:

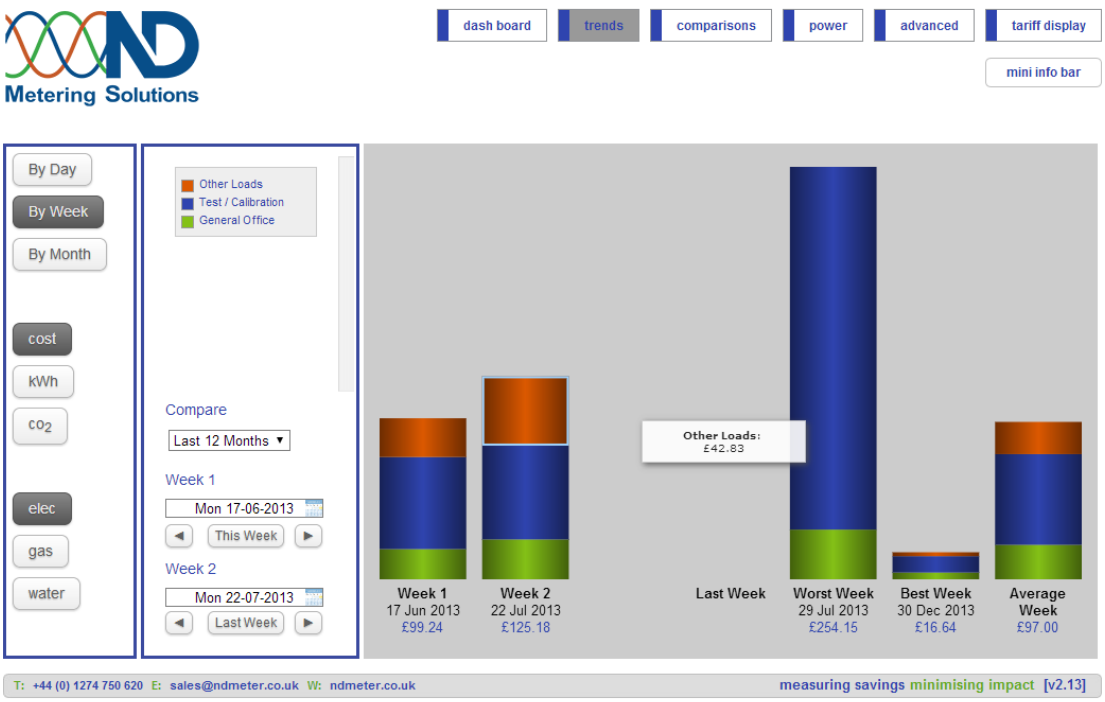
- StruxureWare od Schneider Electric je software zaměřený především pro velké společnosti, který umožňuje z dat generovat různá hlášení, podporuje alarmy, atd.,
- ND Meter od ND Metering je jednodušší aplikací s podporou sledování více měření, umožňující jejich srovnání, sledování trendů, nebo přepočet na cenu,
- Energomonitor je zaměřen více na domácí uživatele, minimalistické rozhraní poskytující informace o spotřebě s výpočtem nákladů za spotřebované energie,
- Engage Platform od efergy, aplikace podobná Energomonitoru, která poskytuje přehled o aktuální spotřebě i spotřebě za poslední den, měsíc a rok, umožňuje hlídat náklady a přepočítává spotřebu na cenu,
- Emoncms, open-source webová aplikace pro zaznamenávání, zpracování a vizualizaci energií, teploty a dalších dat,
- SolarMonitor je aplikace, která umožňuje sledovat výrobu a efektivitu výroby elektrické energie ze solárních panelů. Dokáže generovat jednoduché přehledy, ať už denní, týdenní, měsíční, nebo roční a poskytuje informace o stavu přístrojů, které jsou do procesu výroby zapojeny.

Některými z výše uvedených aplikací se budu zabývat o něco málo podrobněji v následujících odstavcích. Ne pro všechny aplikace je bohužel k dispozici funkční demo, což je případ StruxureWare, u něhož je k dispozici pouze popis funkcí s obrázky z rozhraní.

2.2.1 ND Metering

Společnost Northern Design Electronics Ltd. dodává měřící přístroje s instalací na DIN lištu, nebo panel, pro měření spotřeby energie včetně měření jalového výkonu s možností připojení pomocí rozhraní RS-485, nebo Ethernet. Změnu nastavení přístroje a monitorování aktuálních hodnot lze provádět v administraci přístroje přes webové rozhraní (viz Northern Design (Electronics) Ltd. (2009)). Pro detailní monitorování více přístrojů pak slouží webová aplikace. K dispozici je také funkční demo[3] této aplikace pro potencionální zákazníky. Struktura jednotlivých částí aplikace je pak následující:

[3]Dostupné on-line z: `https://www.energiaccount.com/TRIAL`



Obrázek 2.1: ND Meter Comparisons

Dashboard poskytuje přehled všech elektroměrů, spotřebu za poslední den, týden, nebo měsíc, porovnání aktuálního a předchozího týdne, případně dne, co do spotřebované energie, nebo ceny za tuto energii.

Trends zobrazuje trendy spotřeby, nejhorší, nejlepší a průměrný týden co do spotřeby a ceny vedle vybraných dvou týdnů (standardně aktuální a předchozí týden). Je možné porovnávat nejen týdne, ale také dny a měsíce.

Comparisons slouží ke srovnání dvou různých vybraných období, opět podle spotřeby, nebo ceny (viz obrázek 2.1).

Power zobrazuje aktuální data spotřeby, účiníku, napětí a proudu z vybraných elektroměrů.

Energy Map je grafickou reprezentací spotřeby za zvolené období. Jedná se o tepelnou mapu, kde je nízká spotřeba reprezentována modrou barvou a vysoká spotřeba barvou červenou.

Tariff Display zobrazuje tarifní tabulky pro jednotlivé dny a hodiny s možností zvolit barvu a cenu za spotřebovanou kWh pro různá tarifní pásma, která jsou hodinám dnů přiřazena.

2.2.2 Schneider Electric StruxureWare

Společnost Schneider Electric vyvíjí produkty pro energetický management řady PowerLogic. Jedná se o měřící zařízení v kategorii od základních elektroměrů s instalací na DIN lištu, přes pokročilé panelové měřiče až po špičkové přístroje s instalací do racku.

Pro porovnání jsem vybral řadu PowerLogic PM5500, která dle manuálu (viz Schneider Electric (2013)) umožňuje měřit a zaznamenávat činný i jalový výkon, špičkové odběry, okamžité hodnoty spotřeby na třífázové soustavě i kvalitu odebírané energie. Měřící přístroje této řady umožňují připojení pomocí rozhraní RS-485, nebo Ethernet a jednoduché napojení na software StruxureWare Power Monitoring Expert od Schneider Electric. Zaznamenaná data je však možné využít i v systémech třetích stran.

Demo aplikace[4], které je reprezentováno obrázky ze skutečné aplikace na příkladu univerzitního kampusu, se dělí do sekcí:

Dashboards poskytuje přehled o nákladech za rok v porovnání s rokem předchozím, spotřebu v jednotlivých částech budov, porovnání kvality energie, přehled odebíraného proudu v čase, hodnoty účiníku, porovnání spotřeby a zobrazení spotřeby budov včetně spotřeby jednotlivých pater.

Diagrams slouží pro zobrazení spotřeby a špičkové spotřeby osvětlení, výroby a klimatizace/topení v měřítku jednoho měsíce.

Tables umožňuje zobrazit aktuální data jednotlivých měření v podobě tabulek napětí, proudu, činného výkonu a účiníku, aktuálního napětí a proudu pro jednotlivé fáze, činného a jalového výkonu, a také harmonické složky napětí a proudu, to vše pro jednotlivá měření – budovy.

Alarms představuje seznam abnormálních událostí v měřené síti, například podpětí, přepětí, překročení limitu proudu apod.

Reports generuje tabulky spotřeby energií, ať už měsíční, hodinovou, nebo například po směnách a další. Umožňuje generovat reporty hodnocení kvality energie, systémových hlášení, nebo trendů spotřeby.

2.2.3 Energomonitor

Příkladem další webové aplikace pro energetický management, tentokrát se zaměřením především na domácnosti, je český energomonitor. Společnost energomonitor s. r. o. dodává měřící přístroje, které se instalují do domovních rozvaděčů, a umožňují měřit proud protékající jednotlivými fázemi. Naměřená data jsou rádiově odesílána na přijímač s displejem, kde může zákazník kontrolovat aktuální spotřebu a cenu za spotřebu, a odesílána na servery společnosti.

Uživatel se pomocí svého uživatelského jména a hesla přihlásí na webu společnosti na svou „nástěnku“, kde může kontrolovat a srovnávat spotřebu a výdaje za

[4]Dostupné on-line z: `http://www.powerlogic.com/swpm_demo/`

jednotlivé měsíce, včetně toho aktuálního. Na nástěnce se nachází spotřeba, případně cena za spotřebu, a to pro:

- *Trvale zapnuto*, tedy nejmenší měřenou okamžitou spotřebu tvořenou neustále zapnutými spotřebiči,
- *Průběžná spotřeba*, tvořená spotřebou nad úroveň *Trvale zapnuto*
- a *Celková spotřeba*

Pro výše uvedené kategorie je dále počítána spotřeba za měsíc a denní průměr, obojí děleno na spotřebu v nízkém a vysokém tarifu, pokud má uživatel sazbu s nízkým tarifem, a také procentuální podíl na spotřebě a průměrný příkon.

2.2.4 Shrnutí

StruxureWare od Schneider Electric je z vyjmenovaných produktů bezpochyby nejpropracovanější software, určený zejména velkým podnikům. Má obrovské možnosti výstupu, ať už grafického, v podobě grafů, nebo textového, v podobě tabulek.

ND Meter má jednodušší uživatelské rozhraní a nepodporuje tolik funkcí, jako StruxureWare, přesto ale poskytuje dostatečné možnosti výstupu pro srovnání dat spotřeby energií a umožňuje sledovat více měřících přístrojů.

Nejjednodušší v porovnání s výše uvedenými produkty je energomonitor, který se nezaměřuje na firmy, ale domácnosti. Umožňuje sledovat pouze spotřebu elektrické energie (konkurenti umí například zpracovávat také data spotřeby plynu) a to v co nejjednodušším možném rozhraní. Výstup je omezen na tabulku a graf přehledu s možností srovnání dvou měsíců.

3. Cíl práce

Cílem této práce je návrh a implementace moderní webové aplikace pro energetický management, která by měla uživateli poskytnout možnost snadno a jednoduše prohlížet a analyzovat data spotřeby elektrické energie zachycená přístroji společnosti KMB systems s. r. o. Jelikož je taková aplikace pro rozsah této práce až příliš komplexní, bude funkcionalita ve srovnání se zmíněnými produkty konkurenčních společností omezená.

Požadavkem na technologie je využití ASP.NET MVC na straně serveru, a tedy cílení na platformu Microsoft Windows a webový server IIS. Data jsou získávána prostřednictvím SQL CLR knihovny uložených procedur AKD, jejímž autorem je Bc. Pavel Polívka (2013), z databázového serveru Microsoft SQL Server 2012.

3.1 Funkce

Primární funkcí aplikace je zpracování surových dat měření energií uložených v databázi software ENVIS ve smyslu přepočtu na reálnou energii a případnou agregaci. Data měření budou uživateli reprezentována ve formě grafů spotřeby za uživatelem zvolené období. Je žádoucí, aby mohl uživatel porovnávat spotřeby energií za zvolené období pro všechna zvolená měření, případně porovnávat data jednoho (nebo více) měření v různých časových obdobích, například aktuální a minulý měsíc, apod. Mělo by být možné zobrazovat nejen spotřebu energií, ale také vypočíst cenu za tuto energii podle tarifu uloženého v databázi ENVIS.

Mezi další požadavky na aplikaci patří možnost tvořit z měření stromovou strukturu, což aktuální verze databáze ENVIS neumožňuje. V principu se jedná o možnost každému měření přiřadit jiné měření, jehož je „potomkem“. Měření, které nebude mít žádného předka, bude představovat vrchol stromu. V modelové situaci by se mohlo jednat například o měřící přístroj připojený k hlavnímu jističi budovy. Pod toto měření pak mohou spadat měření na jednotlivých patrech, které budou jeho potomky.

Další přidanou funkcionalitou by měla být možnost přiřadit kterémukoliv měření libovolné množství tzv. štítků – tagů. Podle těchto štítků by bylo možné vyhledávat měření, která spolu souvisí, například data spotřeby pro oddělení, které je rozděleno do několika budov.

4. Návrh a řešení

Jelikož bylo jedním z požadavků na aplikaci využít moderních trendů, rozhodl jsem se pro vývoj aplikace typu SPA. Serverová část aplikace bude obsahovat model, představovaný třídami mapovanými na objekty uložené v databázi ENVIS, a bude poskytovat API pro klientskou část aplikace. Klientská část aplikace bude reagovat na akce uživatele, bude ze serveru pomocí API získávat požadovaná data modelu a ta vizualizovat.

4.1 Schéma databáze software ENVIS

Ze všeho nejdříve je nutné se seznámit se strukturou databáze, kterou vizualizační software ENVIS generuje. Vlastním zkoumáním a konzultacemi s vedoucím práce a Ing. Pavlem Štěpánem jsem zjistil, že mezi nejdůležitější tabulky, ze kterých budu data dotazovat, jsou následující.

SmpObjectDB
: Uchovává místa, ve kterých probíhají měření v podobě identifikátoru objektu a názvu objektu.

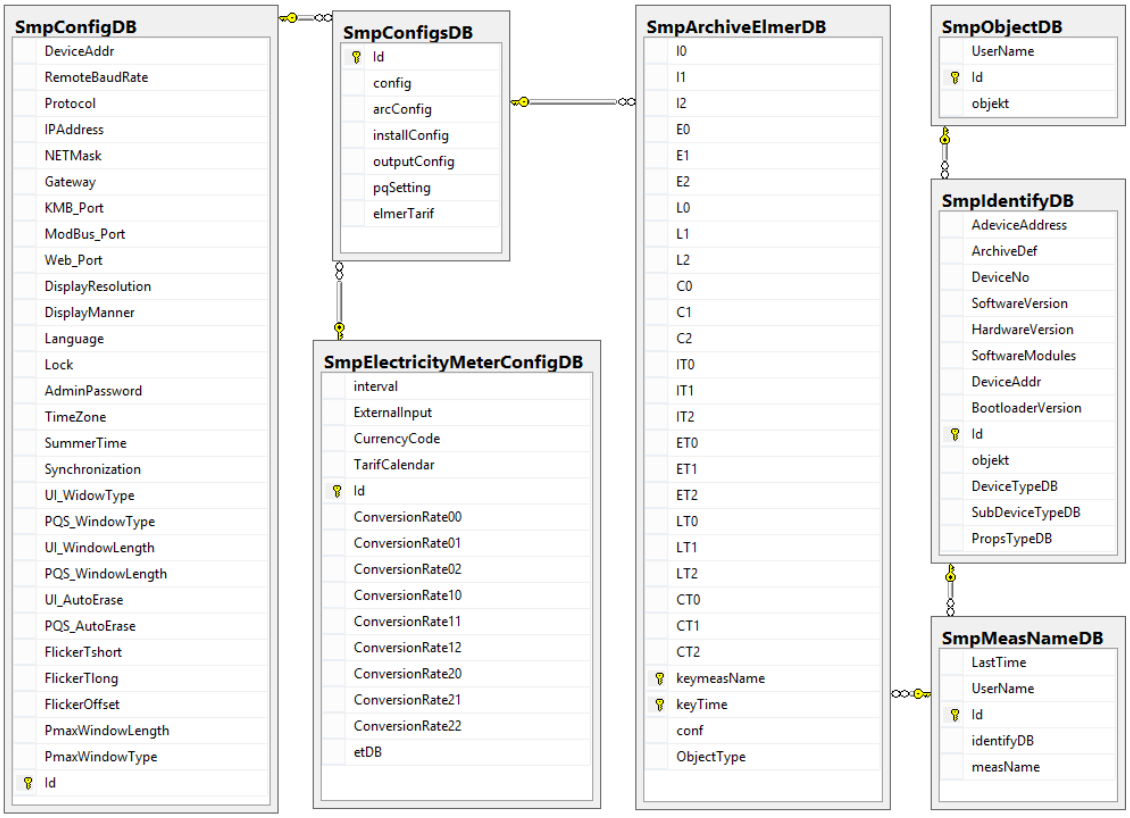
SmpIdentifyDB
: Uchovává záznamy o jednotlivých měřicích přístrojích, například identifikátor *Id* (primární klíč), typ zařízení *DeviceTypeDB*, *SubDeviceTypeDB* a *PropsTypeDB*, verzi firmware a identifikátor objektu z *SmpObjectDB*, ve kterém je přístroj umístěn.

SmpMeasNameDB
: Uchovává informace o jednotlivých měřeních v podobě identifikátoru měření, identifikátoru přístroje z *SmpIdentifyDB*, který měření provádí a názvu měření.

SmpArchiveElmerDB
: Uchovává záznamy z elektroměru pro jednotlivá měření. Každý záznam obsahuje identifikátor měření z *SmpMeasNameDB*, čas měření, konfiguraci přístroje a zejména pak vlastní data měření, která představují hodnotu na elektroměru v době měření. Uchovány jsou činný výkon (Import I a Export E) a jalový výkon (induktivní L a kapacitní C) pro tři fáze soustavy, což nám dává sloupce *I0* až *I2*, *E0* až *E2*, *L0* až *L2* a *C0* až *C2*. Záznam také obsahuje hodnoty souhrnné spotřeby každé ze čtyř měřených složek pro až tři tarify, které jsou

reprezentovány sloupci *IT0*, až *IT2* (pro Import v tarifech 1 až 3), stejně tak *E*, *L* a *C*.
Nutno podotknout, že veškerá data měření jsou uložena v bezrozměrných jednotkách a je nutné je nejprve vynásobit konstantami danými typem přístroje a jeho konfigurací, abychom dostali údaje v kWh pro činný, resp. kVArh pro jalový výkon.

SmpElectricityMeterConfig
Uchovává informace o tarifu v podobě měny, ve které probíhá fakturace, kalendáře, a až třech tarifních tabulek o třech tarifech. Hodnotou z tarifní tabulky je nutné pronásobit spotřebu v kWh, abychom získali cenu za spotřebovanou energii v dané měně. Tarif je vázán na konfiguraci každého záznamu v *SmpArchiveElmerDB*.



Obrázek 4.1: ERD diagram vybraných tabulek databáze ENVIS

4.2 Zdroj dat

4.2.1 AKD

K získání jednotlivých dat měření jsem měl využít knihovnu uložených procedur AKD. Součástí této knihovny je také SQL CLR knihovna, kde jsou některé z procedur implementovány nikoliv v jazyce T-SQL, ale C#, a jsou kompilovány do dynamicky linkované knihovny, která je nahrána na Microsoft SQL Server. Součástí AKD je instalátor, který na zvolený server a databázi knihovnu nahraje a uložené procedury vytvoří.

Pro získání základních energií uložených v tabulce *SmpArchiveElmerDB* slouží procedura *akd_energy_consumption_list* (viz ukázka 4.1). Tato procedura přebírá parametry identifikátoru měření `@keymeasName`, počáteční datum `@startTime`, koncové datum `@endTime` a nakonec parametr `@args`, který určuje jaké z měřených složek budou součástí výsledku volání procedury.

```
CREATE PROCEDURE akd_energy_consumption_list
  @keymeasName int,
  @startTime datetime,
  @endTime datetime,
  @args nvarchar(60)
AS
  EXTERNAL NAME AKD.StoredProcedures.akd_energy_consumption_list
GO
```

Ukázka kódu 4.1: Deklarace procedury *akd_energy_consumption_list*

Parametr `@args` obsahuje názvy měřených složek oddělené čárkami. Činný import – I a export – E, jalová induktivní – L a kapacitní – C a také souhrnné hodnoty těchto složek pro jednotlivé tarify, tedy IT, ET, LT a CT. Na velikosti písmen nezáleží. Pokud například chceme, aby součástí výsledků byla naměřená data složek Import a Export, předáme `'i,e'`, jak uvádí dokumentace AKD (Polívka, 2013, str. 25). Výsledkem volání této procedury jsou zpracovaná data elektroměru s reálnými hodnotami energií pro požadované složky.

4.2.2 Problémy s výkonem AKD

Parametr `@args` pro omezení složek ve výsledcích byl zaveden z důvodu vysoké časové náročnosti přepočtu hodnot energií na hodnoty reálné. Procedura vracela jen ty sloupce, které byly v parametru `@args` specifikovány a zbylé sloupce nebyly součástí výsledku, což se v průběhu návrhu modelu ukázalo jako problém, jelikož proceduru vracející dynamický počet sloupců v závislosti na hodnotě parametru nebylo jednoduše možné mapovat na třídy a volání procedury tak, aby vždy vracela celou sadu sloupců, trvalo neúměrně dlouho.

Vzhledem k problémům s nedostatečným výkonem pro potřeby ve webové aplikaci jsem se rozhodl pro vytvoření pomocné tabulky, do které by byla ukládána data již zpracovaná pomocí AKD a samotná aplikace by data dotazovala právě z této tabulky. Tato tabulka byla nutná nejen z výkonnostních důvodů, ale také proto, že nad výsledky vrácenými uloženou procedurou nebylo možné provádět seskupování ani další agregační funkce.

Další nevýhodou procedury je, že provádí součet hodnot na jednotlivých fázích pro měřené energie, což se po konzultacích s vedoucím práce ukázalo jako nežádoucí a dostal jsem přístup ke zdrojovým kódům knihovny procedur, abych mohl provést úpravy nutné k zachování naměřených dat každé energie i pro jednotlivé fáze.

Optimalizace

Při zkoumání zdrojového kódu procedury *akd_energy_consumption_list* se ukázalo, že procedura provede dotaz na záznamy elektroměru vyhovující danému rozsahu data, a poté ve smyčce pro každý záznam spouští další sérii dotazů pro zjištění časové zóny a letního času, a také konfigurace měřícího přístroje a konstant pro přepočet měřených dat na reálnou energii. Přepsal jsem proceduru tak, aby tyto informace dotazovala předem a uchovávala v poli sloužícím jako lookup tabulka, čímž došlo k podstatnému zrychlení, jak ukazuje tabulka 4.1.

Tabulka 4.1: Porovnání výkonu procedury *akd_energy_consumption_list* před a po optimalizaci

	Před optimalizací		Po optimalizaci	
	Průměr	σ	Průměr	σ
Kompletní sada dat	214,599 s	2,909 s	4,152 s	0,388 s
Data za jeden týden	15,809 s	0,675 s	0,073 s	0,029 s

Metodika testování

Kompletní sada dat představuje celkem 9.422 záznamů elektroměru v čase od 8. ledna 2013 do 23. dubna 2013. Před optimalizací bylo nad touto sadou dat provedeno 9 měření, po optimalizaci pak 20 měření. *Data za jeden týden* představují 673 záznamů v čase od 16. do 23. dubna 2013, počet měření je 20 před i po optimalizaci.

Test byl tvořen iterací kódu z přílohy B.1. Před každým voláním procedury bylo provedeno vyčištění cache pro uložené procedury a bufferů, následně byla spuštěna procedura, a po jejím skončení byla z pohledu *sys.dm_exec_procedure_stats* získána doba trvání. Tato doba byla ukládána do dočasné tabulky, ze které byla na konci procesu pomocí agregačních funkcí *AVG* a *STDEV* získána průměrná hodnota a směrodatná odchylka.

4.3 Změny schéma databáze

Pro jisté odstínění aplikace od databáze generované softwarem ENVIS a pro standardizaci názvů sloupců v rámci aplikace byly vytvořeny pohledy nad některými tabulkami. Důležité bylo zachovat všechny generované tabulky beze změn a nová data ukládat do nových tabulek, pro případ aktualizace schématu softwarem ENVIS.

4.3.1 Tabulky

Instalační skript aplikace přidává do databáze následující tabulky (zde uvedeny bez prefixu *EDW_*).

Variables
: Ačkoliv byl výkonnostní nárůst u procedury *akd_energy_consumption_list* po optimalizaci obrovský, ukázala se procedura jakožto přímý zdroj dat nevhodná, a proto byla celkově přepsána a je použita pro generování obsahu této cache tabulky. Kromě přepočtu naměřených dat na reálné energie, které jsou ukládány přírůstkově, provádí také výpočet rozdílu proti předchozímu měření.

  Každý záznam měření obsahuje hodnoty všech energií pro všechny fáze. Pro čtyři měřené energie a tři fáze elektrické soustavy to představuje dvanáct sloupců. Jelikož bude v budoucnu množství sloupců narůstat, rozhodl jsem se jednotlivé hodnoty každé energie a fáze ukládat do řádku, namísto dvanácti sloupců tedy máme dvanáct řádků.

  Tabulka obsahuje primární klíč *MeasurementId* (který je zároveň cizím klíčem odkazujícím na měření *SmpMeasNameDB*), čas měření *Time*, typ měřené energie *Type* a číslo fáze *Line*. Uchován je také identifikátor sazby *PlanId*, přírůstková hodnota v době odečtu *Value* a rozdílová hodnota proti předchozímu záznamu měření *Consumption*.

TariffConsumptions
: Druhá cache tabulka, která obsahuje souhrnná data spotřeby (sečteny všechny hodnoty na všech fázích) v jednotlivých tarifech pro každou z měřených energií. Primární klíč je podobný tabulce předchozí, *MeasurementId*, čas měření *Time*, typ měřené energie *Type* a identifikátor tarifní tabulky *Tariff*. Dále je opět uchován také identifikátor sazby *PlanId*, přírůstková hodnota v době odečtu *Value* a rozdílová hodnota proti předchozímu záznamu měření *Consumption*.

Measurement_Tags
: Slouží pro uchování štítků jednotlivých měření. Každé měření může mít přiřazeno libovolné množství štítků, podle kterých je možné měření sdružovat a vyhledávat. Primárním klíčem jsou oba sloupce *MeasurementId* a *Tag*, který představuje popisek štítku.

Measurement_Parents
: Slouží pro uchování dvojice měření a jeho předek. Měření může mít přiřazeno

rodiče, pod kterého spadá. Tímto způsobem budou implementována podružná měření pro analýzu ztrát. Primárním klíčem jsou oba sloupce *MeasurementId* a *ParentId*.

4.3.2 Pohledy

Devices
: Pohled nad tabulkou *SmpIdentifyDB* slouží pro standardizaci názvů jednotlivých sloupců a přidává *NULL* sloupec *Name*, který v původní tabulce není obsažen a bude obsahovat název měřícího přístroje načtený za běhu aplikace pomocí dynamicky linkované knihovny KMB-Lib.

Measurement
: Představuje jednotlivá měření z tabulky *SmpMeasNameDB* a připojuje také tabulku *Measurement_Parent*, čímž je ke každému měření doplněn také sloupec rodiče.

Records
: Představuje jeden okamžik měření. K jednomu okamžiku měření jsou pak přiřazeny všechny měřené energie na všech fázích z tabulky *Variables* a také spotřeby v jednotlivých tarifech z tabulky *TariffConsumptions*.

4.4 Serverová část

Jak již bylo řečeno, aplikace poběží na platformě Microsoft Windows pod webovým serverem IIS a data bude získávat z Microsoft SQL Server 2012. Serverová část aplikace funguje na platformě ASP.NET MVC 4 pod .NET Framework.

Jedna z nejdůležitějších částí serverové části aplikace bude API pro klientskou část. Server nebude provádět generování pohledů, nebo dokonce celých stránek, bude pouze poskytovat HTML soubory pohledů, které budou zpracovávány lokálně ve webovém prohlížeči klientskou částí aplikace. Výhodou je „tenčí“ server, který bude mít na starosti poskytování koncových bodů rozhraní API, dotazování dat z databáze na základě požadavků klientské části a jejich zpracování do formy vhodné pro přenos ke klientovi, což znamená především serializaci do formátu JSON.

4.4.1 Model

Nejdříve bylo nutné namodelovat třídy, které budou reprezentovat data uložená v databázi. Jelikož pracuji s databází již navrženou, odpovídá model, až na změny zahrnující přidané pohledy a nové tabulky, způsobu uchování dat v této databázi.

Place
: Představuje umístění, například budovu. V databázi jsou objekty tohoto typu uloženy v tabulce *SmpObjectDB*.

Plan
Představuje tarifní informace uložené v tabulce *SmpElectricityMeterConfigDB*.

Device
Představuje měřící přístroj uložený pod pohledem *Devices* v tabulce *SmpIdentifyDB*. Každý přístroj má přiřazeno umístění *Place*, pod které spadá.

Tag
Představuje štítek měření uložený v tabulce *Measurement_Tags*. Štítek může být přiřazen k více měření a stejně tak měření může mít přiřazeno více štítků.

Measurement
Představuje prováděné měření uložené pod pohledem *Measurement* v tabulce *SmpMeasNameDB*. Měření může mít přiřazeno rodičovské měření *Parent* a kolekci podružných měření *Children*, obojí typu *Measurement*, díky připojené tabulce *Measurement_Parent*. Měření tak mohou představovat stromovou strukturu. Každé měření má přiřazen přístroj *Device* stejnojmenného typu, který měření provádí. Dále může mít měření přiřazenu kolekci štítků *Tags* typu *Tag*.

Record
Představuje jeden záznam měření uložený pod pohledem *Records* v tabulce *Variables*. V jeden okamžik jsou pro dané měření zachyceny všechny měřené energie v jednotlivých fázích a také souhrnné hodnoty energií pro jednotlivé tarify. Záznamu měření je přiřazeno měření *Measurement* a sazba *Plan*, obojí stejnojmenného typu, a obsahuje kolekce *Variables* typu *Variable* a *TariffConsumptions* typu *TariffConsumption*.

Variable
Představuje hodnotu jedné energie a jedné fáze pro jeden záznam měření uložený v tabulce *Variables*. Objekt má přiřazen záznam měření *Record* stejnojmenného typu.

TariffConsumption
Představuje souhrnnou hodnotu jedné energie, tzn. součet hodnot na všech fázích, pro jeden tarif *Plan*. Stejně jako *Variable* má přiřazen záznam měření *Record*.

4.4.2 Mapování dat databáze na model

Pro mapování dat z databáze na třídy modelu byl využit Entity Framework 6. Ten umožňuje z existujícího modelu (tříd) vytvořit databázi, tzv. Code-First přístup, případně z databáze vytvořit model. Bylo využito přístupu Code-First pro již existující databázi, a to z toho důvodu, že model generovaný z již existující databáze by do aplikace zanesl velké množství nepotřebných tříd s původním pojmenováním sloupců a tabulek. Výhodou EF je, že umožňuje dotazovat entity pomocí jazyka

LINQ, který je velice podobný SQL. Dotazy jazyka LINQ je možné dále řetězit a teprve při potřebě získat samotná data se složí SQL dotaz pro databázový server.

V relačních databázích se k vyjádření vztahu mezi entitami používají cizí klíče, v případě objektů jsou tyto cizí klíče reprezentovány referencemi. V terminologii Entity Framework říkáme těmto referencím navigation properties – navigační property.

Konfigurace mapování

Pro konfiguraci mapování jednotlivých entit se používá Fluent API (MSDN Data Developer Center), díky kterému pro každou entitu určíme tabulku, do které se bude entita mapovat, primární klíč entity, způsob generování primárního klíče, jaké atributy se budou mapovat, a případně na jaké sloupce (standardně použije EF pro jméno sloupce název atributu), a pak vztahy mezi dalšími entitami. Kardinalita vztahu může být stejně jako u relačních databázi typu 1:1, 1:N, nebo M:N.

Příklad takové konfigurace je v ukázce B.2. Jedná se o konfiguraci mapování entity měření *Measurement*, kde primárním klíčem je atribut *MeasurementId* (u něhož je zakázáno generování identifikátoru databázovým systémem). Entita je mapována na tabulku, resp. pohled *EDW_Measurement* a mapovány jsou čtyři sloupce: identifikátor *MeasurementId*, identifikátor rodiče *ParentId*, sloužící jako cizí klíč do stejné tabulky, identifikátor zařízení *DeviceId*, sloužící jako cizí klíč do tabulky, resp. pohledu *EDW_Devices*, a jméno měření *Name*.

Nakonec jsou konfigurovány relace. Atribut *Device* je navigačním atributem pro zařízení, a jako cizí klíč se použije atribut *DeviceId*. V tomto případě se jedná z pohledu *Device* o relaci 1:N, kdy jeden přístroj může mít přiřazeno více měření a každé měření pak musí nutně mít přiřazen přístroj, který měření provádí. Dalším atributem je volitelný *Parent*. Ten má více podružných měření v atributu *Children*, který je kolekcí typu *Measurement*. Cizím klíčem pro tento vztah je atribut *ParentId*.

4.4.3 Návrh datového rozhraní

Pro implementaci rozhraní API jsem zvolil knihovnu BreezeJS. Ta umožňuje „zrcadlení“ serverové části modelu, tedy jednotlivých entit, do klientské části aplikace. Aby byly zachovány informace o datových typech atributů entit, poskytuje server klientovi tzv. Metadata, kde jsou tyto informace uloženy, stejně jako například navigační property, tedy vztahy mezi entitami. Klient může dotazovat server pomocí rozhraní implementujícího Open Data Protocol, které umožňuje filtrování dat, řazení, nebo stránkování.

V době návrhu byl BreezeJS provázán s Entity Framework, proto jsem zvolil právě ten a nikoliv například NHibernate. Nyní již BreezeJS podporu pro NHibernate přidal[1], ač stále ve stádiu testování. Bez použití některého z podporovaných ORM frameworků by byl programátor nucen ručně sestavit metadata, která jsou pro dotazování třeba.

[1]Seznam změn dostupný On-line z: `http://www.breezejs.com/documentation/download#148`

Koncové body rozhraní

Rozhraní pro přístup k datům je dostupné na relativní adrese `/api` pod protokolem HTTP a obsahuje dále zmíněné koncové body. Metodou *GET* je z těchto bodů možné získat všechna data, nebo je dle specifikace OData protokolu dále filtrovat. Modifikace ani odstraňování není podporováno. Formát dat je možné ovlivnit HTTP hlavičkou *Accept*, klientská aplikace však bude využívat data ve formátu JSON.

- `GET /api/EDW/Metadata` vrací metadata modelu,
- `GET /api/EDW/Measurements` vrací jednotlivá měření,
- `GET /api/EDW/Devices` vrací jednotlivé měřící přístroje,
- `GET /api/EDW/Places` vrací jednotlivá umístění – objekty,
- `GET /api/EDW/Tags` vrací jednotlivé štítky a přiřazené měření,
- `GET /api/EDW/Plans` vrací jednotlivé sazby a tarifní tabulky,
- `GET /api/EDW/Records` vrací jednotlivé záznamy měření.

Výstup rozhraní

Jak vypadá výstup rozhraní při přístupu k bodu `/api/EDW/Measurements`, si můžete prohlédnout v příloze B.3, která reprezentuje tři měření: *PC1*, *PC2* a *II. patro*, jehož jsou měření *PC1* a *PC2* potomci. Ti mají *II. patro* jako rodiče.

4.4.4 Problém seskupování dat

Jelikož jsou záznamy dat spotřeby elektrické energie pořizovány každých 15 minut (a tento interval se v budoucnu může ještě zkrátit), představuje i jen jeden den, rovných 96 záznamů typu *Record*, každý o čtyřech měřených energiích na jednotlivých fázích, celkem tedy 12 entit typu *Variable*, a dále souhrnná spotřeba těchto energií pro jednotlivé tarifní tabulky sazby, tedy dalších 12 entit typu *TariffConsumption*, po serializaci velké množství dat.

Řešením tohoto problému je seskupení jednotlivých záznamů na straně serveru, a to po hodinách, dnech, týdnech, měsících a rocích. Problém je, že protokol OData ve své třetí verzi nijak nespecifikuje možnost seskupování záznamů (Open Data Protocol, 2014) a knihovna BreezeJS žádnou vlastní implementaci neobsahuje.

Návrh řešení

Možným řešením bylo vytvořit několik koncových bodů rozhraní API, na kterých budou dostupné seskupené záznamy, například `/api/EDW/RecordsGroupedByDays` a další. Jelikož by to pro každé seskupení znamenalo nový koncový bod datového rozhraní, zvolil jsem místo toho variantu jediného bodu `/api/EDW/RecordsGrouped`

s parametrem *GroupingOptions*, který je výčtovým typem s hodnotami `None`, `Hour`, `Day`, `Week`, `Month` a `Year`.

Seskupená data měla být původně získána z databáze díky pohledům, což se záhy ukázalo jako problém, jelikož Entity Framework nedovoluje mapovat jednu třídu na více tabulek (pohled se pro EF tváří jako tabulka). Jelikož musí koncové body rozhraní API kvůli OData vracet data typu `IQueryable`, zvolil jsem nakonec jazyk LINQ pro sepsání dotazů, které data seskupují.

Implementace

Jelikož pro seskupování nelze využít již existujících tříd *Variable* a *TariffConsumption*, a to právě z důvodu omezení mapování jedné třídy na jednu tabulku, byly vytvořeny ekvivalentní třídy *GroupedVariable*, představující seskupený záznam o spotřebě energie, a *GroupedTariffConsumption*, představující seskupený souhrn spotřeby energií pro jednotlivé tarify sazby.

V jazyce LINQ byly implementovány dotazy pro seskupení dat obou tříd po hodinách, dnech, týdnech, měsících a rocích. Dále byla implementována metoda `GetRecordsGrouped` s parametrem `GroupingOptions`, která v závislosti na hodnotě parametru využívá některou z metod seskupení uvedených dotazů a vrací seskupený záznam měření *GroupedRecord*, kam jsou vloženy seskupené záznamy o spotřebě energií a o spotřebě energií pro jednotlivé tarify.

Variantu dotazu pro seskupení spotřeby energií po hodinách si můžete prohlédnout v příloze B.4. Jednotlivé záznamy jsou seskupovány vždy podle identifikátoru měření, pod které spadají, fáze, na které jsou měřeny, typu energie a identifikátoru sazby, dále pak podle času, v tomto případě ořízntutého o hodiny.

4.5 Klientská část aplikace

Aplikace na straně klienta, tzn. webového prohlížeče, by měla mít přívětivé uživatelské rozhraní a celkově by měla být velice jednoduchá, a na rozdíl od software ENVIS, který poskytuje široké spektrum funkcí, se bude zabývat pouze vizualizacemi energií.

4.5.1 Výběr technologií

Prvním problémem byl výběr technologií. Některé z nich jsou již dány, například BreezeJS, jiné bylo nutné nejdříve zvolit.

Jelikož budou součástí výstupu aplikace také grafy, bylo nutné zvolit některou z knihoven pro generování grafů. Na počátku jsem uvažoval o Google Charts, toto API je však umístěno na serverech Google v Internetu, což by mohlo znamenat nefunkčnost u pracovních stanic v intranetu, které přístup k Internetu nemají. Z toho důvodu bylo nakonec zvoleno offline řešení v podobě knihovny Highcharts.

Jelikož je celá aplikace od počátku pojata jako SPA, nemá server na starosti „rendering“ stránky, pouze poskytuje data, která se u klienta vykreslují, v tomto případě prostřednictvím rozhraní API. Vše ostatní musí řešit vlastní aplikace na

straně klienta, tzn. v prohlížeči. Mezi takové věci patří typicky routování, které jinak provádí server, zpracování událostí, asynchronní komunikace se serverem, atd.

Komunikace se serverem

Pro získávání dat ze serveru využívám knihovnu BreezeJS. Ta umožňuje dotazovat entity ze serveru pomocí připraveného rozhraní API, cachovat je, i dotazy, pomocí kterých byly získány. Základem je tzv. *EntityManager*, pomocí kterého vyřizujeme dotazy, *EntityQuery*.

```
var manager = breeze.EntityManager('api/EDW');
var query = breeze.EntityQuery()
  .from('Measurements')
  .where('MeasurementId', 'eq', '1');

manager.executeQuery(query)
  .then(querySucceed)
  .fail(queryFailed);
```

Ukázka kódu 4.2: Dotaz na získání měření s identifikátorem

Jednoduchý příklad dotazu na měření s identifikátorem `1`, které máme uloženo na serveru, si můžete prohlédnout v ukázce 4.2. Funkce `executeQuery` vrací tzv. promise – příslib. Její vykonání je asynchronní, zbytek kódu se provádí okamžitě a na výsledky se nečeká. Příslib nám umožní pomocí funkce `then` předat callback funkci, která bude zavolána, jakmile je dotaz proveden a výsledky, které tento callback obdrží, vráceny. Funkce `fail` se zavolá, pokud při vykonávání dotazu dojde k chybě. BreezeJS využívá přísliby implementované v knihovně Q[2].

BreezeJS samozřejmě umožňuje skládat komplikovanější dotazy (bre) s více podmínkami. Výsledky je možné také řadit, nebo stránkovat. Pomocí navigation properties je možné načítat související entity, atd.

SPA framework

Pro vývoj aplikací typu SPA existuje množství frameworků, které vývoj těchto aplikací ulehčují. Řeší routování, přináší do vývoje některý z návrhových vzorů, ať už je to například MVC, nebo MVVM, s čímž souvisí například data binding, nebo načítání šablon (pohledů).

Při výběru frameworku jsem volil mezi dnes populárním frameworkem Angular a frameworkem Durandal. Bohužel nepříliš srozumitelná oficiální dokumentace a pomalá učící křivka Angularu měly za následek volbu frameworku Durandal.

Samotný Durandal pak má závislosti v knihovně KnockoutJS pro data binding, se kterou souvisí návrhový vzor MVVM, a dále v knihovnách jQuery a RequireJS,

[2]Dokumentace dostupná on-line z: `https://github.com/kriskowal/q/blob/v1/README.md`, [cit. 2014-04-04]

která přináší dělení zdrojového kódu do modulů a umožňuje dynamické načítání modulů a injekci jejich závislostí.

Návrhový vzor MVVM

Návrhový vzor Model View ViewModel je znám především vývojářům, kteří programují aplikace nad Windows Presentation Foundation. Tento vzor vychází z návrhového vzoru Presentation Model (Fowler, 2004).

Model představuje data, se kterými aplikace manipuluje. Data jsou v modelu reprezentována objekty a model poskytuje metody pro manipulaci s nimi. Fyzické umístění dat je pro zbytek aplikace transparentní, aplikace pouze využívá prostředků, které poskytuje model a model se stará o načítání a ukládání dat do úložiště, ať už se jedná o lokální soubor, nebo vzdálený databázový server.

ViewModel obsahuje data, která budou zobrazena v pohledu, a implementuje funkce uživatelského rozhraní. Pokud bude pohledem například formulář s tlačítkem pro odeslání, pak bude ViewModel obsahovat atributy pro hodnoty jednotlivých polí formuláře a funkci pro jeho zpracování po kliknutí na tlačítko odeslání.

View nebo-li pohled, představuje uživatelské rozhraní. Zobrazuje data, která jsou uložená ve ViewModelu, volá jeho funkce a aktualizuje se, pokud se ViewModel změní.

Data Binding

Knihovna KnockoutJS, na které je závislý framework Durandal, je použita pro vytvoření vazby mezi pohledem, představovaným HTML šablonou, a ViewModelem. Pokud uživatel v pohledu změní hodnotu prvku, který je vázán na ViewModel, provede Knockout aktualizaci ViewModelu. Aby bylo možné provést aktualizaci pohledu při změně ViewModelu, obalují se hodnoty atributů do zvláštních objektů, tzv. observables, díky kterým dokáže Knockout změny detekovat, viz článek na toto téma, který napsal García (2013).

AMD moduly

Veškerý JavaScript kód je obalen do tzv. AMD modulů. Jednotlivé moduly mohou mít specifikovány závislosti, což jsou další moduly, případně knihovny třetích stran. Knihovna RequireJS zajišťuje asynchronní načítání všech závislostí používaných modulů, navíc ve správném pořadí. Moduly mohou být typu singleton (vždy vrací jeden objekt), nebo může vracet objekt zastávající funkci konstruktoru. V takovém případě je možné mít více instancí jednoho modulu.

```
define(['jquery', 'knockout'], function ($, ko) {

  var viewModel = {
    firstName: ko.observable('John'),
    lastName: ko.observable('Doe')
  };

  viewModel.activate = function() {
    // Durandal callback volaný před kompozicí modulu do stránky.
  };

  return viewModel;
});
```

Ukázka kódu 4.3: Jednoduchý AMD modul vracející ViewModel

V ukázce 4.3 si můžete prohlédnout jednoduchý modul typu singleton, který vrací objekt reprezentující ViewModel. Prvním argumentem funkce `define` je pole závislostí, které daný modul má, druhým je funkce reprezentující modul, která má jako parametry právě definované závislosti.

UI framework

Při prvotním návrhu jsem uvažoval nad budováním uživatelského rozhraní a jeho prvků, tedy například tlačítek, prvků formuláře, ovládacích prvků, jako jsou záložky, kontejnerové prvky, apod., zcela od základů. Z důvodů časové náročnosti a nutnosti pečlivého ladění v prohlížečích jsem však od tohoto ustoupil a zvolil prototyping framework Bootstrap v jeho třetí verzi.

Bootstrap implementuje responzivní grid systém, pomocí kterého lze obsah dělit do sloupců a buněk s proměnnou šířkou. Pomocí různých CSS tříd lze buňkám mřížky nastavit viditelnost pro některá zařízení, jako jsou počítače, tablety, nebo mobilní telefony, a přeskupovat tak obsah stránky podle toho, ve kterém zařízení je prohlížena (boo). Bootstrap navíc implementuje množství ovládacích prvků, kontejnerových prvků a utility tříd pro snazší návrh uživatelského rozhraní.

4.5.2 Struktura aplikace

Struktura celé aplikace vychází z konvencí frameworku Durandal. Aplikace se nachází ve složce `/App`, v níž jsou umístěny podsložky:

- `/modals` obsahuje moduly dialogových oken,
- `/model` obsahuje komponenty aplikace,
- `/utils` obsahuje moduly s pomocnými funkcemi,
- `/views` obsahuje HTML šablony představující pohledy aplikace,

- `/viewmodels` obsahuje moduly ViewModelů vázaných na pohledy,
- `/services` obsahuje moduly služeb aplikace, například pro komunikaci se serverem, načítání a zpracování dat modelu,
- `/main.js` je spouštěcí skript aplikace,
- `/setup.js` je modul starající se o zavedení vlastních data binding handlerů a rozšíření prototypů některých JavaScriptových objektů.

Základem aplikace je tzv. *shell*, zvláštní ViewModel volaný spouštěcím skriptem, ve kterém probíhá například inicializace routingu. K němu je vázán pohled, který představuje obal aplikace, tedy hlavičku aplikace s logem, hlavní menu, patičku a zvláštní `div`, do kterého je vložena aktivní stránka. Stránkou myslíme pohled a k němu přiřazený ViewModel.

Každý ViewModel může definovat zvláštní funkce, které Durandal volá v průběhu kompozice stránky. Jejich seznam se nachází v dokumentaci frameworku, viz Eisenberg (a). Tyto funkce, které dokumentace označuje jako callbacky, jsou frameworkem volány při různých událostech kompozice, například když je ViewModel připojen k DOM dokumentu.

Router

Při volání callbacku `activate` modulu *shell* se provádí inicializace routeru aplikace. Pomocí routeru je zpracována část URL adresy za hashem (znak `#`), která je porovnávána s definovanými routami. Pokud je nalezena routa vyhovující adrese, provede se načtení modulu přiřazeného k dané routě a jeho kompozice do *shell*.

Jednotlivé routy mohou být parametrizované. Součástí adresy je jeden nebo více parametrů, které jsou modulu předány pomocí callback funkcí. Parametry mohou být volitelné, nebo povinné. Konfiguraci routeru aplikace si můžete prohlédnout v ukázce 4.4.

```
var routes = [
  { route: '', moduleId: 'overview', title: 'Přehled' },
  { route: 'overview*subpage', hash: '#overview/', moduleId: 'overview',
    title: 'Přehled', nav: true },
  { route: 'comparison', moduleId: 'comparison', title: 'Srovnání',
    nav: true },
  { route: 'places', moduleId: 'places', title: 'Místa' }
];

return router.makeRelative({ moduleId: 'viewmodels' })
  .map(routes)
  .buildNavigationModel()
  .activate();
```

Ukázka kódu 4.4: Zjednodušená inicializace routeru aplikace

V konfiguraci routeru v předešlé ukázce se nachází pole objektů jednotlivých rout. Atribut *route* je použit jako maska pro porovnávání s URL adresou, *hash* slouží pro vytváření odkazů a je generován automaticky, pokud jej neurčíme. *moduleId* je cesta k modulu – ViewModelu, který bude pro vyhovující routu načten, *title* je text, kterým bude nahrazen titulkem dokumentu, a nakonec *nav* udává, jestli bude daná routa součástí navigace, kterou z takto konfigurovaného routeru v pohledu generujeme v podobě menu. Volání `makeRelative`, kde předáváme objekt s atributem *moduleId*, nastavuje relativní cestu k jednotlivým modulům definovaným v routách.

O možnostech parametrických rout, nebo routerech potomků (implementace submenu), si můžete přečíst v dokumentaci, viz Eisenberg (b).

Služby aplikace

Pro manipulaci s daty modelu jsou některé funkce implementovány kategoricky do modulů tzv. služeb, a to za účelem udržení kódu ViewModelů prostého od složitější logiky pro zpracování dat.

datacontext poskytuje instanci *EntityManager* a tzv. lookup objekt, který se stahuje při spuštění aplikace, a který obsahuje tarify, umístění, měření, tagy a přístroje, jakožto data, která se používají často.

ChartService poskytuje funkce pro zpracování dat záznamů *Record*, nebo *GroupedRecord*, a získání objektu konfigurace pro knihovnu Highcharts, která vygeneruje reprezentaci těchto dat v podobě grafů. V době psaní textu jsou implementovány funkce pro generování grafů křivek energií pro srovnání měření a sloupcových grafů pro vizualizaci spotřeby měření.

logger poskytuje funkce pro výpis a logování událostí, ať už chybových nebo informativních. V případě potřeby může být událost zobrazena v uživatelském rozhraní, jinak jsou události pouze zaznamenávány do konzole prohlížeče.

lookups poskytuje funkce pro získání umístění, měření, a jiných dat z lookup objektu služby `datacontext` podle identifikátoru.

plan poskytuje funkci pro přepočet energie na cenu dle dané sazby a aktivního tarifu.

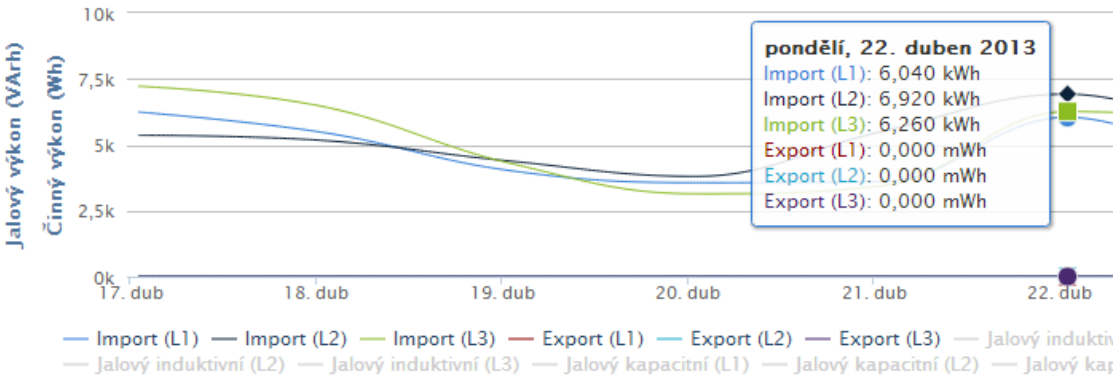
records poskytuje funkce pro získání záznamů měření pro daný identifikátor měření (případně všechna měření), seskupení a rozsah data.

4.6 Funkce a jejich implementace

V této kapitole se budu zabývat popisem některých z komponent aplikace a uživatelským rozhraním aplikace spolu s funkcemi, které aplikace poskytuje.

4.6.1 Implementace grafů

Grafy jsou jedním z primárních výstupů aplikace a budou se vyskytovat na více místech. Pro samotné vykreslování grafů byla použita knihovna Highcharts, která je zaobalena ve znovupoužitelném modulu – komponentě. Pro nastavení vykreslení a dat je knihovně předáván zvláštní `options` objekt (hig).



Obrázek 4.2: Ukázka generovaného grafu energií

Zpracování dat

Pro vygenerování objektu s nastavením slouží modul `services/ChartService`, který implementujte funkce přebírající data (pole záznamů *Record* nebo *GroupedRecord*) a objekt s nastavením, kterým bude přetíženo základní nastavení (nepovinné). Služba poskytuje následující funkce:

`getAreaChartOptions`

Slouží pro vytvoření plošného grafu pro jednotlivé energie měření se sloučenými popisky.

Pro každou energii uloženou v entitě *Variable* nebo *GroupedVariable* je vytvořena série, do které jsou přidávány jednotlivé hodnoty dané energie.

`getStackedColumnChartOptions`

Slouží pro vytvoření sloupcového grafu spotřeby v jednotlivých tarifech. Tento typ grafu je používán v přehledech pro vizualizaci podílů spotřeby. Sloupec, představující den, je složen ze třech částí, představující tarify.

Nejdříve se provádí seskupení záznamů *Record* podle počátečního data *LocalStartTime* funkcí `_.groupBy` knihovny UnderscoreJS, která vrací „slovník“ s poli reprezentujícími skupiny záznamů se shodným počátečním datem.

Výsledné skupiny představují v grafu vždy jeden sloupec pro dané datum. Pro každý záznam ve skupině je zpracováno pole spotřeby v tarifech *Tariff-Consumptions* a jsou sečteny hodnoty těch záznamů energií činného výkonu (Import), jejichž tarif je stejný. Tím vzniknou pro každý sloupec tři hodnoty spotřeby pro tři tarify, ze kterých je vytvořeno pole pro sérii grafu.

Součástí funkce je objekt se základním nastavením, tj. předpřipravené osy, nastaven správný typ grafu, tloušťka čar, nebo funkce pro formátování popisků sérií a jednotlivých os. Objekt základního nastavení může být přepsán, případně rozšířen pomocí volitelného parametru, k čemuž je využito funkce `$.extend` s volbou tzv. „hluboké“ kopie. Nakonec je do objektu nastavení vloženo pole jednotlivých sérií, které jsme z dat vytvořili a objekt je vrácen.

Implementace

Vlastní implementace se nachází v modulu `model/chart/Chart`, který vrací funkci konstruktoru. Parametrem konstruktoru je objekt s nastavením pro knihovnu Highcharts (hig). Tento je uchován v observable atributu `options`.

Při připojení pohledu modulu do DOM je vytvořena instance grafu voláním `Highcharts.Chart`. Pokud dojde k aktualizaci `options` objektu, dojde k aktualizaci grafu. Jednotlivé série grafu jsou porovnávány s novým objektem nastavení na základě jejich jmen. Data těch existujících jsou aktualizovány, nové jsou do grafu přidány.

Pokud je třeba indikovat načítání dat pro graf, jsou k dispozici funkce pro částečné překrytí grafu zprávou o stahování dat.

4.6.2 Tabulka přehledu spotřeby

Tabulka zobrazující přehled spotřeby a ceny za spotřebu používaná v sekci *Přehledu* a *Měření* umožňuje pomocí parametru `options` sestavit tabulku s variabilním počtem řádků, popisky a formátováním pro data spotřeby v určitém časovém rozsahu.

V závislosti na nastavení `options` objektu je možné generovat tabulku s přehledem spotřeby pro zvolené měření, případně jako součet spotřeb všech měření v databázi. V horní části tabulky, jak ukazuje obrázek 4.3, se nachází přepínání mezi zobrazením spotřeby a ceny za spotřebu, která je vypočtena dle sazby přiřazené danému měření.

Implementace

Tabulka je implementována v modulu `model/tables/StatisticsTable`, který vrací funkci konstruktoru tabulky. Ten jako argument přebírá objekt nastavení, viz ukázka v příloze B.5.

Atribut *view* objektu nastavení udává, zda bude standardně zobrazována spotřeba (0), nebo cena za spotřebu (1), zatímco *measurement* obsahuje identifikátor měření, pro které budou data načítána. Hodnota `False` znamená součet všech měření. Následuje pole jednotlivých řádků tabulky, kde *label* představuje popisek, *from*

Obrázek 4.3: Tabulka přehledu spotřeby

a *to* rozsah data, pro který budou data načítána, *format* je způsob formátování rozsahu data, který je zobrazen pod popiskem, *type* je pole proměnných, jejichž součet bude tvořit zobrazovanou hodnotu a nakonec *units* jsou základní jednotky pro zobrazovanou hodnotu.

Při inicializaci modulu dochází k dotázání serveru na data pro jednotlivé řádky tabulky definované v objektu nastavení. Jednotlivé záznamy vrácené serverem jsou zpracovány, vybrané složky energií jsou filtrovány podle atributu *type* objektu nastavení a jejich hodnoty sečteny, a to jak složky spotřeby, tak ceny za spotřebu. Zpracovaná data jsou vložena do sloupce hodnot tabulky.

Pro formátování hodnot a jednotek je využita služba `services/units`. Pokud bude zobrazovaná hodnota například 1.100.000 a jednotky jsou nastaveny na Wh, zobrazí se v tabulce hodnota 1,100 MWh (standardně formátováno na tři desetinná místa), jak je vidět na obrázku 4.3.

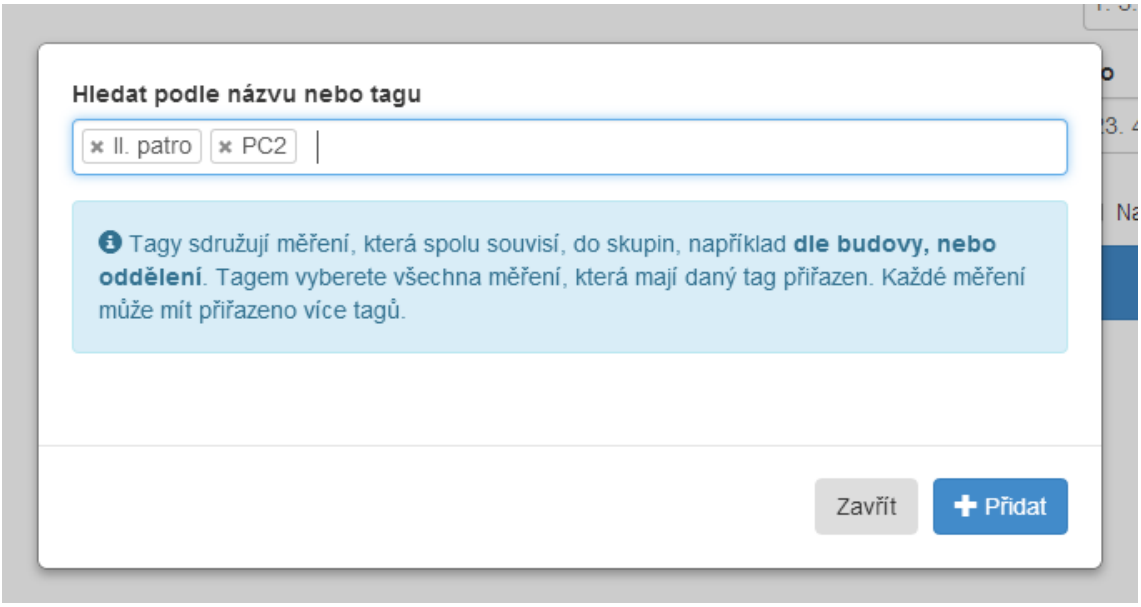
4.6.3 Dialog výběru měření

Další z komponent je modální dialogové okno pro výběr měření, které je využito například v sekci Srovnání spotřeby (viz 4.6.6), a které si můžete prohlédnout na obrázku 4.4.

Dialogové okno obsahuje vstupní pole, které se po nakliknutí rozevře do podoby seznamu štítků a měření. Psaním je možné filtrovat obě kategorie a kliknutím na položku je tato přidána na seznam vybraných, jak je patrné na obrázku 4.4.

Tlačítko *Přidat* uzavře dialogové okno a vrátí pole identifikátorů vybraných měření, tlačítko *Zavřít* pak vrací prázdné pole. Pokud je vybrán štítek, je nahrazen identifikátory všech měření, které jej mají přiřazen.

Implementace dialogového okna se nachází v modulu `modals/measurements`. Při připojení pohledu do DOM je sestaveno pole štítků a měření z `lookups` objektu

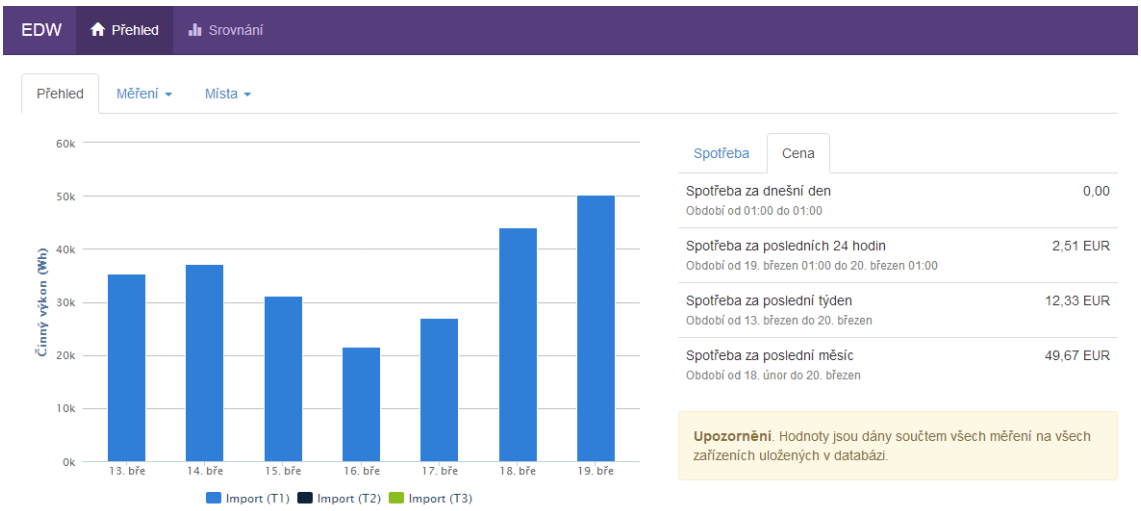


Obrázek 4.4: Hledání měření podle názvu nebo štítku

služby `datacontext` a je inicializován prvek výběrového seznamu, který je těmito daty naplněn.

4.6.4 Přehled

Sekce *Přehled* je vstupním bodem aplikace a poskytuje základní informace o souhrnné spotřebě energií všech měření. Na obrázku 4.5 si můžete všimnout grafu, který reprezentuje souhrnnou spotřebu v jednotlivých tarifech za uplynulý týden.



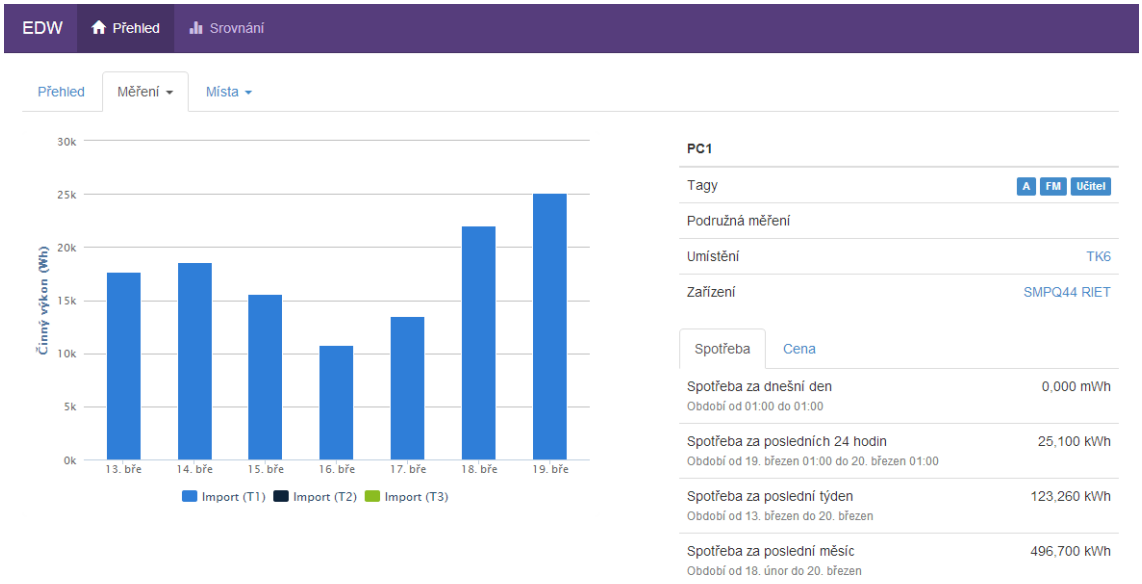
Obrázek 4.5: Sekce *Přehled*

Vpravo se nachází tabulka, která obsahuje informace o spotřebě za aktuální den, uplynulých 24 hodin, týden a měsíc, s možností zobrazení ceny za spotřebu. Tato sekce je tvořena následujícími znovupoužitelnými komponentami:

- `model/charts/Chart`, obalující modul knihovny Highcharts, který umožňuje aktualizaci sérií grafu po inicializaci,
- `model/tables/StatisticsTable`, modul dynamické tabulky.

4.6.5 Měření

Sekce *Měření* je přehledem pro konkrétní měření, a proto je prakticky identická se sekcí *Přehled*, viz obrázek 4.6. Kromě grafu spotřeby v jednotlivých tarifech za poslední týden a tabulky spotřeby, obsahuje navíc tabulku s názvem měření, typem měřícího přístroje, seznam přiřazených štítků a podružných měření.

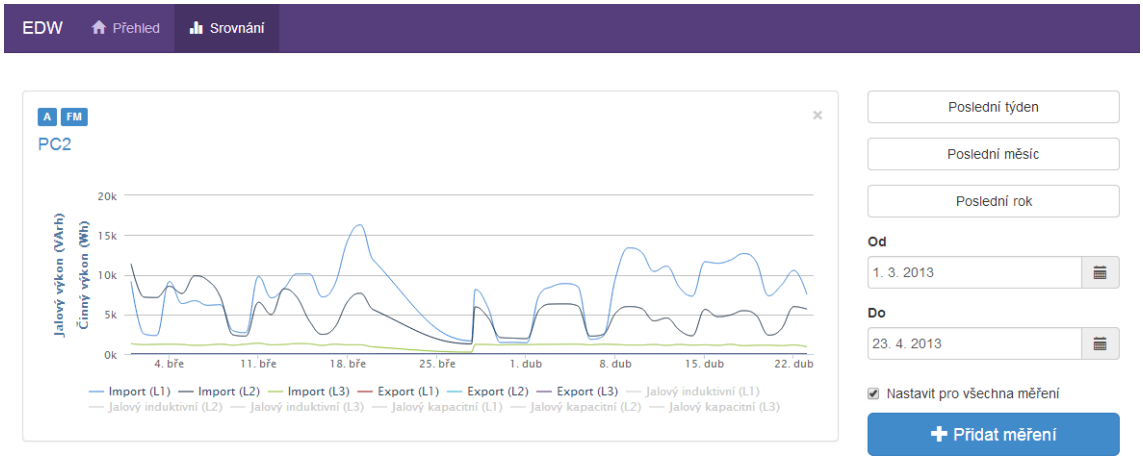


Obrázek 4.6: Sekce *Měření*

4.6.6 Srovnání spotřeby

Sekce *Srovnání* umožňuje srovnat vybraná měření v nastaveném rozsahu data, případně pro každé srovnávané měření nastavit vlastní rozsah. Jelikož může být jedno měření přidáno i vícekrát, je možné srovnávat trendy spotřeby v čase.

Na obrázku 4.7 si můžete všimnout ovládacích prvků pro nastavení rozsahu data na pravé straně. Rozsah je možné změnit pomocí tlačítek s předpřipravenými volbami, nebo ručně, pomocí dvou polí pro datum. Standardně je vyplněn rozsah data jednoho týdne zpět od posledního záznamu v databázi. Díky komponentám kalendáře pro každý ze vstupů může uživatel rozsah jednoduše změnit.



Obrázek 4.7: Sekce *Srovnání*

Tlačítko *Přidat měření* otevře dialog výběru měření (viz 4.6.3), kterým je možné do srovnání přidat libovolná měření. Nutno podotknout, že každé přidané měření má svůj vlastní graf. Je to z toho důvodu, že každý záznam měření obsahuje čtyři měřené typy energií (činný výkon, export, jalová induktivní a jalová kapacitní, z čehož jsou po přidání zobrazovány jen první dva typy) pro tři fáze. Pokud by bylo více měření umístěno do jednoho grafu, byl by tento graf velice nepřehledný, ačkoliv není vyloučeno, že v budoucnu nedojde v tomto směru ke změně.

Všechna přidaná měření jsou zobrazována vlevo vedle ovládacích prvků, a kromě tlačítka křížku (×) pro odstranění měření ze srovnání v pravém horním rohu, standardně neobsahují žádné další ovládací prvky. Teprve pokud není aktivní volba *Nastavit pro všechna měření* v pravém panelu, zobrazí se pro každý graf měření vlastní vstupní prvky pro nastavení rozsahu data a typ seskupení dat.

Při jakékoliv změně rozsahu data, ať už „globálního“, při aktivní volbě *Nastavit pro všechna měření*, nebo pro konkrétní měření, dojde ke stažení nových dat a aktualizaci grafu, případně grafů.

Implementace

Každé měření přidané do srovnání je instancí komponenty reprezentované modulem `model/measurement/MeasurementComparison`. Tento modul pak obsahuje stejně jako sekce *Srovnání* vlastní ovládací prvky pro nastavení rozsahu data a navíc také seskupení.

Modul sekce vytváří instance měření pro srovnání na základě pole identifikátorů vráceného z dialogu pro výběr měření a zajišťuje synchronizaci hodnot rozsahu data při aktivaci nebo deaktivaci volby *Nastavit pro všechna měření*. Při změně hodnoty této volby jsou informovány všechny moduly měření.

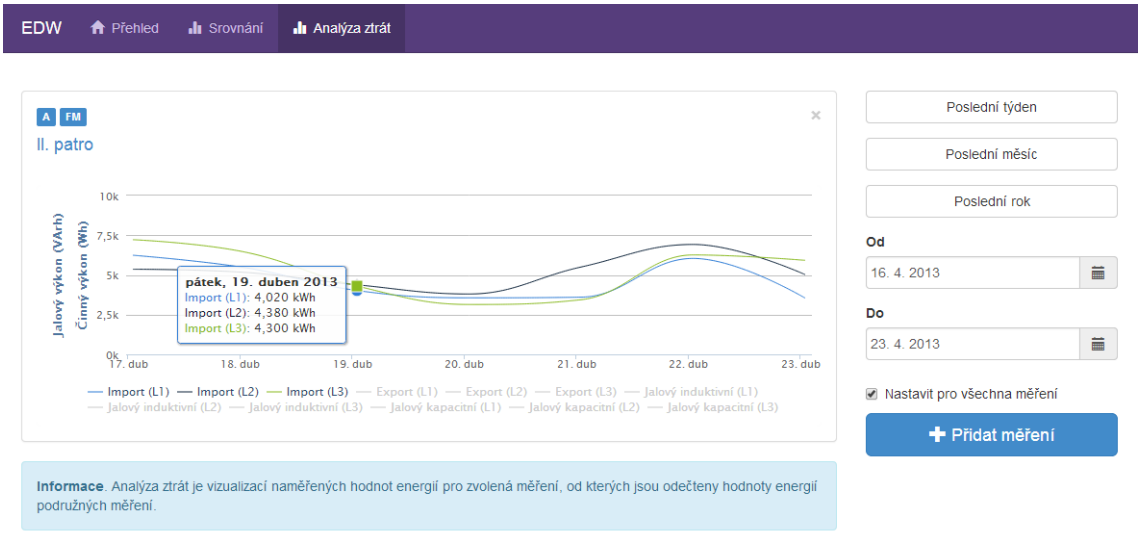
Každý modul měření má na starosti načítání záznamů měření pomocí služby `services/records`, vytvoření objektu nastavení pro modul grafu, tentokrát s využitím služby `services/ChartService`, a nakonec aktualizaci `options` observable

komponenty `model/charts/Chart` (viz 4.6.1). Po dobu dotazování serveru je pak indikováno stahování dat.

4.6.7 Analýza ztrát

V této sekci může uživatel provádět tzv. analýzu ztrát. Sekce je velice podobná sekci *Srovnání*, ovládací prvky jsou stejné, uživatel může do srovnání přidávat více měření, podmínkou ale je, že mají tato měření přiřazeny alespoň jedno podružné měření.

Do grafu pro měření je pak zaznamenána spotřeba energií zvoleného měření, od které jsou odečteny spotřeby energií všech podružných měření. Výsledkem jsou ztráty. Může se jednat o skutečné ztráty, ale také spotřebu neměřených spotřebičů.



Obrázek 4.8: Sekce *Analýza ztrát*

5. Vyhodnocení

5.1 Výkon aplikace

5.1.1 Seskupování záznamů měření

Pro přístup k datům měření jsem měl k dispozici knihovnu AKD. Kvůli problémům s výkonem (viz 4.2.2) jsem byl nucen zavést pomocnou cache tabulku, do které byla ukládána data zpracovaná pomocí AKD. Později, když došlo k přepracování AKD, a ke změnám v modelu kvůli přidání podpory pro data na jednotlivých fázích, sice klesla časová náročnost získání dat, jak ukazuje tabulka 4.1, nebylo však možné provádět nad procedurou vrácenými daty další operace, jako je řazení, seskupování, atd. Navíc bylo potřeba převádět hodnoty elektroměru z přírůstkových na rozdílové, pomocná cache tabulka tedy zůstala.

Všechny dotazy pro seskupování záznamů měření jsou psány v LINQ na straně serveru. Rozhraní API umožňuje pro každý zdroj provádět určité filtrování, řazení, limitování počtu výsledků a podmínky, které jsou připojeny k LINQ dotazu seskupení. Teprve z tohoto celku je složen SQL dotaz, který je odeslán databázovému serveru. Jedná se o tzv. deferred execution.

Jelikož není výsledný generovaný SQL dotaz zcela optimální a vykonávání některých dotazů může na slabších strojích trvat i několik sekund, vyzkoušel jsem i možnost přepsat dotazy provádějící seskupení do T-SQL v podobě table-valued funkcí a jejich výsledky použít pro mapování na entity a jako zdroj pro LINQ dotazy, které vytváří seskupené entity *GroupedRecord*. Ačkoliv bylo seskupování samotné, prováděné pomocí těchto funkcí i při mapování na entity *GroupedVariable* rychlejší, při použití v LINQ dotazu pro sestavení záznamů *GroupedRecord* byl výkon ještě nižší, než v případě hůře vypadajícího SQL dotazu generovaného čistě pomocí LINQ.

Test výkonu API

Provedl jsem několik testů výkonu API v rámci dat, které mám k dispozici. Jedná se o dobu zpracování a seskupování dat měření na straně serveru od zaslání API požadavku po přijetí výsledků.

Jak můžete vidět v tabulce 5.1, seskupování po hodinách je nejpomalejší operací. V reálném nasazení postrádá smysl seskupování po hodinách pro takový rozsah dat a aplikace standardně využívá seskupování po dnech, které je mnohem rychlejší.

Nízký výkon při seskupování po hodinách je dán SQL dotazem pro materializaci entit modelu generovaným z LINQ-to-SQL, při kterém dle exekučního plánu dochází

Tabulka 5.1: Doba zpracování API požadavku na seskupené záznamy měření

Seskupení	Průměrný čas (ms)	Směrodatná odchylka (ms)
Po hodinách	7.114	478
Po dnech	850	57
Po týdnech	2.754	96
Po měsících	3.762	187
Po rocích	450	31

k využití *tempdb*. Generované dotazy totiž kromě seskupování tabulek spotřeby energií *EDW_Variables* a spotřeby v jednotlivých tarifech *EDW_TariffConsumptions*, spojují tabulky a párují jednotlivá měření *EDW_Measurements* a sazby *SmpElectricityMeterConfigDB*.

Řešení

Jako částečné řešení se ukázalo odstranit z třídy *GroupedRecord* reference na měření *Measurement* a sazbu *Plan*, a modifikovat klientskou část, aby tyto načítala z lookup objektu, a zavést nové koncové body API:

- `/api/EDW/VariablesGrouped`, který vrací seskupené záznamy měření obsahující pouze spotřeby energií *Variables*,
- `/api/EDW/TariffConsumptionsGrouped`, který vrací seskupené záznamy měření obsahující pouze spotřebu energií v jednotlivých tarifech *TariffConsumptions*.

Díky tomu se zjednodušil generovaný dotaz, jelikož není nutné připojovat tabulky sazeb a měření, a k tomu seskupené záznamy spotřeby energií, resp. spotřeby energií v jednotlivých tarifech, které nejsou v klientské části vždy využity.

První testy sice zaznamenaly úspěch, výkon seskupení po hodinách však klesl na polovinu. Nakonec došlo k úpravám tabulek spotřeby energií *EDW_Variables* a spotřeby energií v jednotlivých tarifech *EDW_TariffConsumptions*, mapovaných na třídy *Variable* a *TariffConsumption*, do kterých byly přidány sloupce pro předpočítané hodnoty roku, měsíce, data týdne, dne a hodin, podle kterých je seskupováno. Drobnými změnami prošel také skript pro import dat. Po těchto změnách byl znovu změřen výkon.

Jak je patrné z tabulky 5.2, výkon proti stavu před změnou razantně vzrostl, zejména pak seskupování po hodinách, u kterého docházelo k využívání *tempdb* a tím poklesu výkonu. Dosahované časy se samozřejmě pozitivně projevily také na samotné aplikaci, data se rychleji načítají a doba čekání na výstup některých komponent se rapidně zkrátila.

Tabulka 5.2: Doba zpracování API požadavku na seskupené záznamy měření po optimalizaci

Seskupení	Průměrný čas (ms)	Směrodatná odchylka (ms)
Po hodinách	1.002	15
Po dnech	222	4
Po týdnech	194	3
Po měsících	197	11
Po rocích	180	9

Metodika testování

Pro měření hodnot v tomto testu byl použit Apache JMeter. Samotné měření probíhalo ve smyčce nakonfigurované pro 100 opakování pro právě testovaný bod rozhraní API. Při každé iteraci byl odeslán *HTTP Request* s volbou jednoho konkrétního měření a s hlavičkou *Accept* nastavenou na hodnotu *application/json*, ke kterému byl připojen *Graph Results* listener pro sběr dat podle článku o testování webových služeb (Nevedrov, 2006).

Každý dotaz zpracovává celkem 113.064 záznamů spotřeby energií a stejný počet záznamů spotřeby v jednotlivých tarifech sazby měření (před změnou) za dobu od 8. ledna 2013 do 23. dubna 2013. Měřený čas je doba od odeslání HTTP požadavku na koncový bod API po přijetí výsledků.

5.1.2 Nároky rozhraní API na šířku pásma

Jedním z možných problémů pro koncové uživatele je velikost dat přenášených ze serveru na klienta. Každý okamžik měření *Record*, který obsahuje kolekci spotřeby pro jednotlivé energie a fáze *Variables*, případně kolekci spotřeby energií v jednotlivých tarifech sazby *TariffConsumptions*, je serializován do formátu JSON.

V závislosti na zvoleném rozsahu dat a seskupení může přenášený balík dat dosahovat i desítek MiB[1], které server musí odeslat a klient stáhnout. V prostředí Internetu neznamená uplink serveru zpravidla omezení, jelikož je vyšší, než je downlink klienta, takže ačkoliv se podařilo optimalizovat výkon rozhraní API, může být aplikace brzděna malou šířkou pásma klienta při stahování dat měření.

Řešení

Řešením tohoto problému je zapnutí komprese přenášených dat pro webový server IIS. V konfiguraci IIS je nutné zapnout kompresi jak pro statické soubory, tak zejména pro dynamický obsah, kterým data přenášená přes rozhraní API jsou. Pouhé

[1]Balík dat přenášený v průběhu testu API (viz 5.1.1) při seskupení po hodinách měl velikost 9,3 MiB.

zapnutí dynamické komprese však nefunguje, a jak píše Hanselman (2011), je třeba ještě dodatečně nakonfigurovat IIS, aby komprimoval také typ *application/json; charset=utf-8*. V případě IIS 8 bylo ještě nutné nastavit úroveň komprese dynamického obsahu *system.webServer/httpCompression/dynamicCompressionLevel*.

Tabulka 5.3: Porovnání velikosti komprimovaných a nekomprimovaných dat z rozhraní API

	Úroveň komprese		Kompresní poměr
	GZIP L2	Vypnuto	
Metadata popisující entity modelu	2.648 B	15.219 B	82,6 %
Objekt *lookups* obsahující umístění, přístroje, měření, štítky a sazby	1.025 B	2.639 B	61,1 %
Data měření z testu výkonu API seskupená po dnech	24.848 B	414.390 B	94,0 %
Data měření z testu výkonu API seskupená po hodinách	529.616 B	9.787.178 B	94,6 %

Bylo provedeno několik testů za pomoci příkazu `curl` (viz ukázka 5.1) při nastavení dynamické komprese GZIP na úroveň 2 z 10, kde 0 znamená bez komprese a 10 nejvyšší stupeň komprese. Jak je patrné z tabulky 5.3, je komprese textového formátu JSON velice účinná, zejména pak u dat měření, jejichž struktura se opakuje a mění se pouze hodnoty.

```
# curl stáhne data z URL a wc spočte počet bytů
curl -i -H "Accept-Encoding: gzip" URL | wc -c
curl -i URL | wc -c
```

Ukázka kódu 5.1: Získání velikosti komprimovaných a nekomprimovaných dat

6. Závěr

Cílem práce bylo získat informace o aplikacích pro energetický management a získané poznatky využít při návrhu a implementaci vlastní webové aplikace pro základní vizualizaci dat spotřeby elektrické energie, získaných měřícími přístroji a analyzátory kvality elektrické energie od společnosti KMB systems s. r. o. Veškerá data jsou uložena v databázi generované softwarem ENVIS. Aplikace samotná bude fungovat na platformě Microsoft Windows pod běhovým prostředím ASP.NET a .NET Framework 4 a využívat databázový systém Microsoft SQL Server.

Naměřená data měla být získávána pomocí knihovny uložených procedur AKD, její rychlost se však v průběhu vývoje ukázala jako zcela nedostačující a navíc jí nebylo možné přímo použít pro další zpracování získaných dat v rámci dotazu v jazyce T-SQL. Jedna z procedur knihovny, kterou aplikace využívá, byla nakonec optimalizována a přepsána a její výstup je ukládán do pomocné cache tabulky.

Pro samotnou aplikaci, která byla od počátku navrhována jako aplikace typu SPA, bylo navrhnuto a implementováno API rozhraní využívající protokol OData. Pomocí tohoto rozhraní může aplikace načítat například seznam umístění měřicích přístrojů, informace o přístrojích samotných, informace o měřeních a samotná data měření, které je navíc možné seskupovat po hodinách, dnech, týdnech, měsících, nebo letech.

Na základech rozhraní API je postavena aplikace, která poskytuje základní informace o všech měřeních, tabulku přehledu spotřeby a grafy spotřeby, včetně zobrazení ceny za spotřebovanou energii dle konfigurace sazby. Dále pak srovnání jednotlivých měření za účelem vizualizace trendů spotřeby pro různá časová období, případně srovnání spotřeby různých měření a nakonec analýzu ztrát.

Výsledná aplikace je plně funkční, její výstup odpovídá výstupu aplikace ENVIS a představuje základ pro případnou navazující práci, která by měla rozšířit funkcionalitu a množství prezentovaných dat.

6.1 Další rozvoj

Aplikace v tuto chvíli poskytuje pouze základní data a její funkcionalita může být v porovnání s některými zmiňovanými produkty (viz 2.2) omezená. Vysoce komplexní analytický nástroj nebyl cílem této práce, přesto zde prostor pro další vývoj je.

Aplikace byla dosud vyvíjena jako jednouživatelská, obsluhující databázi jednoho podniku, pokud však bude sloužit pro více uživatelů z různých podniků, bude třeba

implementovat správu uživatelů s možností přiřazovat jednotlivým uživatelům práva na prohlížení pouze dat měření z jejich vlastního podniku. Jednotliví uživatelé by také mohli mít různé role, díky kterým by mohli mít různý pohled na stejná data. Například uživatelé z účetního oddělení by viděli primárně cenu za spotřebovanou energii, zatímco energetici především vlastní spotřebu.

Mezi další funkce bychom mohli zařadit generování hlášení, ať už ručně, nebo automatizovaně s rozesíláním na e-mail a výhodou by bylo také více jazykových mutací. Hlavním problémem v tuto chvíli jsou pevně vložené řetězce v HTML šablonách pohledů. Pro řetězce dosazované skrze ViewModel aplikace překlady podporuje pomocí i18n plug-inu knihovny RequireJS.

A nakonec by mohlo být zajímavé propojit tuto jednoduchou webovou aplikaci s aplikací ENVIS tak, aby mohl uživatel stáhnout z webu data, která si právě prohlíží a zobrazit je v ENVIS.

Literatura

Bootstrap Grid System. Dostupné z: http://getbootstrap.com/css/#grid. [online], [cit. 2014-04-08].

Breeze Introduction. Dostupné z: http://www.breezejs.com/documentation/introduction. [online], [cit. 2014-04-04].

Highcharts Options Reference. Dostupné z: http://api.highcharts.com/highcharts. [online], [cit. 2014-04-10].

BERNERS-LEE, T. – CAILLIAU, R. *Proposal for a HyperText Project*, Listopad 1990. Dostupné z: http://www.w3.org/Proposal.html. [online], [cit. 2014-02-27].

EISENBERG, R. *Hooking Composition & Activator Callbacks*, a. Dostupné z: http://durandaljs.com/documentation/Hooking-Lifecycle-Callbacks.html. [online], [cit. 2014-03-31].

EISENBERG, R. *Router*, b. Dostupné z: http://durandaljs.com/documentation/Using-The-Router.html. [online], [cit. 2014-04-03].

FOWLER, M. *Presentation Model*, Červenec 2004. Dostupné z: http://martinfowler.com/eaaDev/PresentationModel.html. [online], [cit. 2014-03-30].

GARCíA, J. G. *Introduction, or How to Enrich Your HTML Views With Unparagoned User Experience*, Duben 2013. [online], [cit. 2014-03-30]. Dostupné z: http://www.barbarianmeetscoding.com/blog/2013/04/09/barbarian-meets-knockout-dot-js-introduction/.

HANSELMAN, S. *Enabling dynamic compression (gzip, deflate) for WCF Data Feeds, OData and other custom services in IIS7*, Březen 2011. Dostupné z: http://www.hanselman.com/blog/EnablingDynamicCompressionGzipDeflateForWCFDataFeedsODataAndOtherCustomServicesInIIS7.aspx. [online], [cit. 2014-04-26].

KMB systems, s. r. o. *SMV, SMP, SMVQ, SMPQ - Multifunkční panelové přístroje a analyzátory kvality energie*, Říjen 2013.

KMB systems s.r.o. *ENVIS Software User Guide for Supported Measuring Instruments Version 1.1*, Září 2012.

MAJDA, F. *Čtvrthodinové maximum. Elektro*. Prosinec 2008, , č. 12, s. 26–28. ISSN 1210-0889.

Mozilla Developer Network. *HTML5*, Únor 2014. Dostupné z: `https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/HTML5`. [online], [cit. 2014-02-27].

MSDN Data Developer Center. *Configuring/Mapping Properties and Types with the Fluent API*. Dostupné z: `http://msdn.microsoft.com/cs-cz/data/jj591617.aspx`. [online], [cit. 2014-04-20].

NEVEDROV, D. *Using JMeter to Performance Test Web Services*, Únor 2006. Dostupné z: `http://www.oracle.com/technetwork/articles/entarch/jmeter-performance-testing-094115.html`. [online], [cit. 2014-04-25].

Northern Design (Electronics) Ltd. *IP Operations Manual*, Srpen 2009. Dostupné z: `http://www.ndmeter.co.uk/files/IP_Operations_Manual.pdf`. [online], [cit. 2014-03-02].

Open Data Protocol. *OData Version 3.0 Core Protocol*, Únor 2014. Dostupné z: `http://www.odata.org/documentation/odata-version-3-0/odata-version-3-0-core-protocol/`.

POLíVKA, P. *Návrh a implementace datové a aplikační vrstvy databáze pro vzdálený přístup, webové aplikace a reporting*. Bakalářská práce, Technická univerzita v Liberci, 2013.

Schneider Electric. *PowerLogic™ PM5500 series User manual*, Prosinec 2013. Dostupné z: `http://download.schneider-electric.com/files?p_File_Id=365018951&p_File_Name=HRB1684301-01.pdf`. [online], [cit. 2014-03-03].

A. Textové přílohy

A.1 Obsah přiloženého CD

- */Deployment* obsahuje instalační balík aplikace,
- */Prerequisites* obsahuje soubory nutné pro instalaci aplikace,
- */Report* obsahuje zdrojové soubory této zprávy pro XeLaTeX,
- */Screenshots* obsahuje obrázky pořízené z aplikace,
- */bp.pdf* je soubor obsahující tuto výstupní zprávu,
- */README* obsahuje název práce a obsah CD.

A.2 Nároky aplikace a instalace

A.2.1 Požadavky

- webový server IIS ve verzi 8,
- Microsoft Web Deploy ve verzi 3.5,
- běhové prostředí .NET Framework ve verzi 4 nebo vyšší,
- ASP.NET MVC 4,
- databázový server Microsoft SQL Server 2012 nebo novější,
- a SQL Server Management Studio.

Aplikace byla testována v prohlížeči Chrome 34 a funguje v nejnovějších verzích všech rozšířených prohlížečů:

- Google Chrome 30 a vyšší,
- Microsoft Internet Explorer 9 a vyšší,
- Mozilla Firefox 27 a vyšší.

Zejména v nižších verzích prohlížeče Microsoft Internet Explorer není možné zaručit bezproblémovou funkčnost, kvůli některým chybějícím funkcím JavaScript interpreteru.

A.2.2 Vytvoření schéma databáze

Pokud máte nainstalován veškerý požadovaný software, vytvořte databázi aplikace pomocí následujícího postupu:

1. Z webových stránek KMB systems s. r. o., v sekci *Ke stažení* → *Software*, si stáhněte software ENVIS ve verzi 1.1,
2. Spusťte software ENVIS, připojte se k databázovému serveru a vytvořte novou databázi,
3. Spusťte SQL Server Management Studio, otevřete a spusťte instalační skript *Prerequisites/Install.sql*,
4. Importujte data měření pomocí ENVIS skrze nabídku *Data* → *Importovat data*.

Nyní bychom měli mít vytvořeno databázové schéma ENVIS včetně pomocných tabulek nutných pro fungování webové aplikace.

A.2.3 Nasazení aplikace

Pokud máte nainstalován Web Deploy 3.5, je nasazení aplikace na server jednoduché.

1. Spusťte Správce internetové informační služby IIS, který by měl být nainstalován spolu s rolí webového serveru IIS,
2. V levém menu rozbalte server a otevřete *Weby*,
3. V pravém menu *Akce* zvolte *Přidat web*, zvolte název, cestu a nastavte vazby dle svých preferencí, volbu *Spustit web ihned* ponechte aktivní,
4. V levém menu rozbalte nově přidaný web, v pravém menu *Nasazení* zvolte *Importovat aplikaci*,
5. Zvolte balíček *Deployment/EDWeb.zip*, pokračujte volbou *Další*,
6. Nastavení přepisování existujících souborů můžete ponechat na přednastavených hodnotách, pokračujte volbou *Další*,
7. *Cesta k aplikaci* nastavte na **prázdný řetězec**, aby aplikace běžela v kořenovém adresáři a *edwContext* connection string upravte podle konfigurace vašeho serveru, pokračujte volbou *Další* a dokončete import.

Ve *Fondech aplikací* nezapomeňte zkontrolovat, že fond (angl. pool), kterému je aplikace přiřazena, má povoleno běhové prostředí .NET Framework 4 nebo vyšší.

A.3 Testovací sestava

Všechny výkonnostní testy byly prováděny na mém stolním počítači v následující konfiguraci.

Procesor
Intel i5-750 @ 3,6 GHz

Operační paměť
8 GB DDR3

Systémový disk
128GB SSD Crucial M4, CT128M4SSD2

Datový disk
1TB Western Digital Caviar Blue, WD10EALS

Operační systém
Microsoft Windows 8.1 (64bitový)

B. Ukázky kódu

```
CREATE TABLE #temp (
  Number INT IDENTITY,
  Elapsed INT,
  PRIMARY KEY (Number)
);

DECLARE @I INT;
SET @I = 0;

WHILE @I < 20
BEGIN

  DBCC FREEPROCCACHE;
  DBCC DROPCLEANBUFFERS;

  -- Parametry @MeaurementId, @StartTime, @EndTime, @Args
  -- Příklad volání:
  EXEC akd_energy_consumption_list 1, '2013-04-16', '2013-04-23', 'i,e,l,c';

  INSERT INTO #temp (Elapsed)
  SELECT last_elapsed_time FROM sys.dm_exec_procedure_stats;

  SET @I = @I + 1;
END;

SELECT
  MAX(Number) AS Iterations,
  AVG(Elapsed) AS Average,
  STDEV(Elapsed) AS StandardDeviation
FROM #temp

DROP TABLE #temp;
```

Ukázka kódu B.1: Kód pro testování rychlosti uložených procedur

```
public class MeasurementMap : EntityTypeConfiguration<Measurement>
{
  public MeasurementMap()
  {
    // Primary Key
    this.HasKey(m => m.MeasurementId);
    this.Property(t => t.MeasurementId)
      .HasDatabaseGeneratedOption(DatabaseGeneratedOption.None);

    // Table & Column Mappings

    this.ToTable("EDW_Measurement");
    this.Property(m => m.MeasurementId);
    this.Property(m => m.ParentId);
    this.Property(m => m.DeviceId);
    this.Property(m => m.Name);

    // Relationships

    // Each measurement references device performing the measurement.
    this.HasRequired(d => d.Device)
      .WithMany(m => m.Measurements)
      .HasForeignKey(d => d.DeviceId);

    // Each measurement can have optional parent.
    this.HasOptional(measurement => measurement.Parent)
      .WithMany(child => child.Children)
      .HasForeignKey(measurement => measurement.ParentId);
  }
}
```

Ukázka kódu B.2: Ukázka konfigurace mapování pomocí Fluent API

```
[
  {
    "$id": "1",
    "$type": "EDWeb.Models.Measurement, EDWeb",
    "Parent": {
      "$id": "2",
      "$type": "EDWeb.Models.Measurement, EDWeb",
      "Parent": null,
      "MeasurementId": 3,
      "ParentId": null,
      "DeviceId": 3,
      "Device": null,
      "Name": "II. patro",
      "Tags": [],
      "Children": [
        {
          "$ref": "1"
        },
        {
          "$id": "3",
          "$type": "EDWeb.Models.Measurement, EDWeb",
          "Parent": {
            "$ref": "2"
          },
          "MeasurementId": 2,
          "ParentId": 3,
          "DeviceId": 2,
          "Device": null,
          "Name": "PC1",
          "Tags": [],
          "Children": []
        }
      ]
    },
    "MeasurementId": 1,
    "ParentId": 3,
    "DeviceId": 1,
    "Device": null,
    "Name": "PC2",
    "Tags": [],
    "Children": []
  },
  {
    "$ref": "3"
  },
  {
    "$ref": "2"
  }
]
```


Ukázka kódu B.3: Ukázka výstupu rozhraní API pro měření

```
public DbQuery<GroupedVariable> VariablesGroupedByDays
{
  get
  {
    var query = from v in Context.Variables
                group v by new
                {
                  // Identifikátor měření
                  MeasurementId = v.MeasurementId,

                  // Předpočítaný DateTime bez hodin
                  Date = v.Date,

                  // Fáze
                  Line = v.Line,

                  // Energie
                  Type = v.Type,

                  // Identifikátor sazby
                  PlanId = v.PlanId
                } into g
                select new GroupedVariable
                {
                  MeasurementId = g.Key.MeasurementId,
                  Line = g.Key.Line,
                  Type = g.Key.Type,

                  PlanId = g.Key.PlanId,

                  StartTime = g.Key.Date,
                  EndTime = g.Max(v => v.Time),

                  LocalStartTime = g.Min(v => v.LocalTime),
                  LocalEndTime = g.Max(v => v.LocalTime),

                  Consumption = g.Sum(v => v.Consumption)
                };

    return (DbQuery<GroupedVariable>)query;
  }
}
```

Ukázka kódu B.4: Seskupení energií *Variable* po dnech

```
{
  view: 1,
  measurement: False,
  rows: [
    {
      label: 'Spotřeba za dnešní den',
      from: time.truncateTime(time.getNow()).toDate(),
      to: time.getNow().toDate(),
      format: 'HH:mm',
      type: ['Import'],
      units: 'Wh'
    },
    {
      label: 'Výroba za dnešní den',
      from: time.truncateTime(time.getNow()).toDate(),
      to: time.getNow().toDate(),
      format: 'HH:mm',
      type: ['Export'],
      units: 'Wh'
    },
    {
      label: 'Jalová induktivní za dnešní den',
      from: time.truncateTime(time.getNow()).toDate(),
      to: time.getNow().toDate(),
      format: 'HH:mm',
      type: ['Inductive'],
      units: 'VArh'
    },
    {
      label: 'Jalová kapacitní za dnešní den',
      from: time.truncateTime(time.getNow()).toDate(),
      to: time.getNow().toDate(),
      format: 'HH:mm',
      type: ['Capacitive'],
      units: 'VArh'
    },
  ]
}

// Pozn.: time je službou utils/time
```

Ukázka kódu B.5: Ukázka objektu nastavení pro tabulku přehledu